(2)

WRDC-TR-89-1118

INTEGRATED CIRCUITS FOR AVIONICS

Raymond Siferd
Wright State University
Dept. of Electrical Engineering
134 Fawcett Hall
Dayton, Ohio 45435

October 1989

Final Report for Period July 1985 – June 1989

Approved for public release; distribution unlimited.

Avionics Laboratory
Wright Research Development Center
Air Force Systems Command
Wright–Patterson Air Force Base, Ohio 45433

[illegible]
RICHARD C. STERLING, GS-13
Project Engineer
Communications Technology Group

Franklin Hutson
FRANKLIN T. HUTSON, GM-14
Chief
Communications Technology Group

FOR THE COMMANDER

[illegible]
CHARLES H. KRUEGER JR
Director
System Avionics Division
Avionics Laboratory

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-89-1118
6a. NAME OF PERFORMING ORGANIZATION Wright State University; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Avionics Laboratory (WRDC/AAAI) Wright Research Development Center
6c. ADDRESS (City, State, and ZIP Code) Dept. of Electrical Engineeirng Wright State University Dayton, Ohio 45435	7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6543
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research & Development Center; 8b. OFFICE SYMBOL (If applicable) WRDC/AAAI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-85-C-1718
8c. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6543	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
62204F	7662	01	45

11. TITLE (Include Security Classification)
Integrated Circuits for Avionics

12. PERSONAL AUTHOR(S)
R. Siferd, J. Birbal, W. Sweet, R. Hohne

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 7/85 TO 6/89	25 October 1989	122

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	01	
25	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
VLSI, Signal Processing, Residue Arithmetic, Narrow Band Filter, CMOS, Vector Processor, Systolic

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The main thrust of the Integrated Circuits for Avionics Program is to calibrate the chip area requirements and performance that can be obtained from custom VLSI circuits which have been designed to accomplish a specific communication signal processing task. The two primary areas of investigation were stringent narrow band filtering based on highly parallel VLSI systolic array architectures and vector processors based on the residue number theory and highly parallel VLSI pipelined architectures. Designs are included for custom VLSI programmable filters in both NMOS and CMOS technologies along with measured performance of higher order filters resulting form cascading the custome prototype chips. A unique multirate sampling scheme is presented for improving filter response. A unique custom VLSI design is included for a high performance pipelined residue processor using 16-bit operands and providing 32-bit results. Measured performance of this architecture is also included based upon testing of a prototype custom VLSI circuit. This program was sponsored by Wright Research & Development Center, WRDC/AAAI, Wright-Patterson AFB, OH 45433-6543.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Richard Sterling	22b. TELEPHONE (Include Area Code) (513) 255-3455; 22c. OFFICE SYMBOL WRDC/AAAI

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Contents

List of Figures

List of Tables

Section 1

INTRODUCTION

This report summarizes the accomplishments and findings under the Integrated Circuits for Avionics program. In this program, Wright State University designed, simulated, obtained fabrication, and tested Very Large Scale Integrated (VLSI) circuits. Each of the VLSI circuits were custom designed to meet specifications provided by the government for a real time signal processing application. The design, simulation, and testing of the VLSI circuits were accomplished at Wright State University with computer aided design tools and special testers assembled as part of the VLSI Design and Test Laboratory. The fabrication was obtained through the DARPA sponsored MOS Implementation Service (MOSIS) [4]. The VLSI mask data developed during the design and simulation phase were submitted to MOSIS over the CSNET and the fabricated circuits were returned to the university in 8–10 weeks. The actual performance of the fabricated custom VLSI circuits was measured to provide a benchmark of performance verses circuit architecture, required chip area, and fabrication technology.

1.1 PROGRAM REQUIREMENTS AND OBJECTIVES

The main thrust of this program is to calibrate the performance that can be obtained from custom VLSI circuits which have been designed to accomplish a specific communication signal processing task. The specific signal processing tasks were defined by the government and were in the areas of stringent narrow band filters based on highly parallel VLSI systolic array architectures and vector processors based on the residue number theory and highly parallel VLSI pipelined architectures. The results are based on actual hardware realization and testing of prototype VLSI circuits. For each of the three tasks, two custom prototype VLSI circuits were delivered along with an interim technical report detailing the design, simulation, and tested performance. Although the circuits were not to be directly integrated into a communication subsystem, the delivered package includes complete details of the architectural approach, circuit logic designs, physical layouts, and simulated and tested performance. This information will significantly improve the capability to specify VLSI architectural approaches for signal processing circuits and also to specify more realistic requirements with regard to performance.

1.2 PROGRAM STRUCTURE

The program consisted of three major tasks:

Task I A Programmable Finite Impulse Response Filter Using Custom NMOS VLSI Components. Included as part of this task was the completion of a Custom VLSI Design and Testing Facility at Wright State University.

Task II A Programmable Digital Filter Using CMOS VLSI Technology. This task differed from Task I in the approach for implementing arithmetic functions for fast adders and multipliers as well as the fabrication technology.

Task III A High–Performance Vector Processor Using Residue Number Theory, Pipelining, and VLSI Technology.

1.3 SUMMARY OF ACCOMPLISHMENTS

Specific accomplishments with quantified results will be discussed under the section for each Task; however, listed below is a summary of several accomplishments resulting from the Integrated Circuits for Avionics Program.

1. Defined unique VLSI architectures for implementing stringent programmable finite impulse narrowband filters.
2. Measured performance and chip area requirements of custom VLSI programmable filters in both NMOS and CMOS technologies.
3. Presented a unique approach to cascading lower order filters to improve performance using multi–rate sampling.
4. Defined a unique VLSI architecture for implementing a high–performance vector processor based on residue number theory.
5. One of the very few hardware VLSI realizations of a high performance pipelined residue vector processor. To the best of our knowledge, the only residue number system processor on a single chip using 16–bit operands and producing 32–bit results.
6. Measured performance and chip area requirements for the high performance pipelined residue vector processor.
7. Calibration of design times, chip area requirements, and performance of typical VLSI signal processing circuits. This will improve cost and performance specifications for such circuits in future developments.
8. Development of a good design, simulation, and test capability for custom and application specific VLSI circuits near the WRDC complex.

1.4 REPORT SUMMARY

This Final Report contains, in addition to this introduction, four technical sections.

Section 2 presents the specification, design, simulation, and tested performance for Task I, a programmable finite impulse response filter using custom NMOS VLSI circuits.

Sections 3 and 4 present information corresponding to Section 2 for Tasks II and III.

Section 5 presents a summary of the findings under the Integrated Circuits for Avionics program and includes recommendations for future work.

Section 2

TASK I - PROGRAMMABLE FINITE IMPULSE RESPONSE FILTER USING CUSTOM NMOS VLSI CIRCUITS

This section describes the design approach and implementation of a custom NMOS VLSI architecture to perform finite impulse response digital filtering. The chip exploits pipelining and parallel processing to increase throughput rates. Individual design components (cells) are presented along with simulation results. System response is fully programmable within the limits imposed by filter length and processing speed. Optimized coefficients are software generated using the Remez Exchange algorithm to minimize error. Performance evaluations of third order and ninth order filter chips are presented. Necessary support circuitry of an efficient programmable filtering system is discussed. This section summarizes the details presented in the Task I Interim Technical Report [1].

2.1 Specifications

The following specifications were stated as design goals:

- The filtering system is to be able to process signals with frequency components in the 0 to 3 MHz range;
- Coefficients are to be selected so as to pass a 25–kHz window out of the 0– to 3–MHz range;
- The system is to be programmable in order to select different bandpass filter responses;
- Out of band rejection of at least 80–dB.

As mentioned above, these are the performance goals of this project. Unfortunately, some of these goals will not be met. First of all, the 80–dB out of band rejection is a high expectation of any filtering system. This goal will not be met when coupled with the small window width of the passband. The desired response of a digital filter is influenced by several factors, the main ones being the amount of passband and stopband ripple that can be tolerated, the width of transition regions, and the width of the bands. In order to have the 25–kHz passband along with acceptably narrow transition regions, a tradeoff needs to be made with the amount of passband and stopband ripple. In effect, this means that the system described hereafter, in order to satisfy passband and transition width goals, will not satisfy the 80–dB out of band rejection goal. However, out of band rejection in the range of 50 to 60 dB may be achieved while meeting passband and transition width requirements by cascading several filters. Further discussion on filter response and the effects of cascading filter chips may be found in Section 2.6.

2.2 FIR Theory

2.2.1 General FIR Filters

Any discrete time system that is capable of accepting a digitized input and applying some operation to it to yield an output sequence that is some modified variation of the input can be classified as a digital filter. The basic block concept of the digital filtering system considered here is shown in Fig. 2.1.

The difference equation for a digital filter can be written in the following general form:

$$y(nT) = \sum_{i=0}^{N} h(i)\, x\,(nT - iT) - \sum_{k=0}^{M+1} b(k) y(nT - kT)\ .$$

This equation relates the nth sample of the output to the N previous values of the output and to the M + 1 most recent values of the input. By choosing T, the sample period, to be one, this equation can be written in a less complex form:

$$y(n) = \sum_{i=0}^{N} h(i)\, x\,(n - i) - \sum_{k=0}^{M+1} b(k) y(n - k)\ .$$

A filter whose present output value is dependent on past output values is said to be recursive in nature. A nonrecursive filter is one in which the output is solely dependent on past input values, i.e., all of the b(k) coefficients are zero. This type of filter is more stable, and desired responses can generally be attained by a filter of reasonable length. Rewriting the above equation for a nonrecursive digital filter yields

$$y(n) = \sum_{i=0}^{N} h(i)\, x\,(n - i)\ .$$

This equation can be realized in direct form as shown in Fig. 2.2.

2.2.2 Linear Phase FIR Filters

A linear phase filter is one with a phase shift which is a linear function of frequency resulting in a constant time delay for all frequency components. For a linear phase filter, the coefficients are symmetrical which permits a realization that is more efficient than the direct form. Since $h(i) = h(N-1-i)$, the implementation of a linear phase filter can be obtained as shown in Fig. 2.3. By taking advantage of the symmetry of coefficients, the proper samples can be added before multiplying by the coefficients. This reduces the number of multiples that must be made from N, for the direct form, to $[(N-1)/2] + 1$ for the linear phase form.

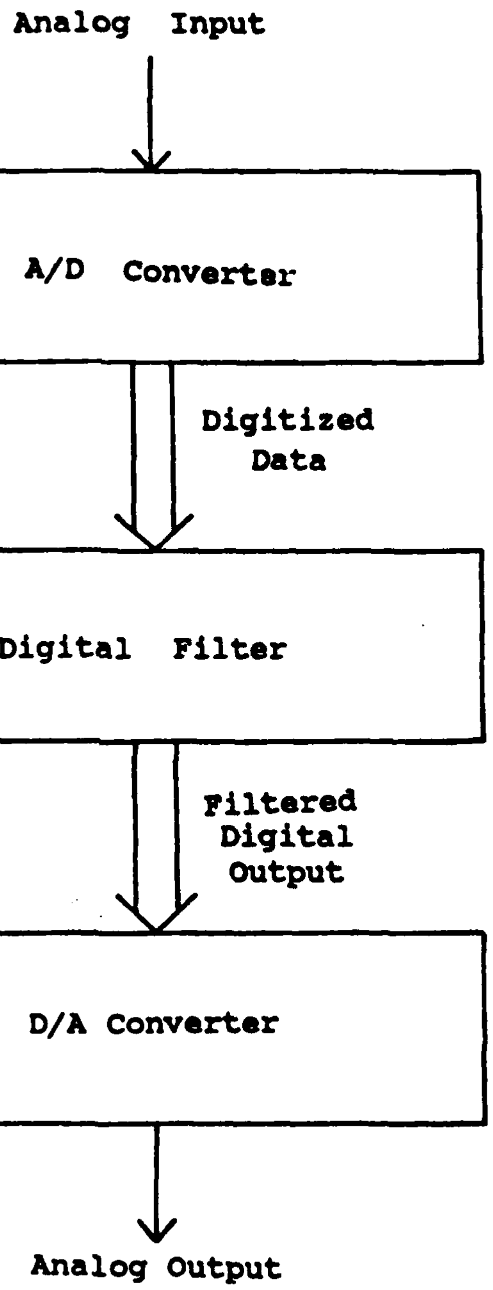


Figure 2.1

Block Diagram of a Digital Filtering System

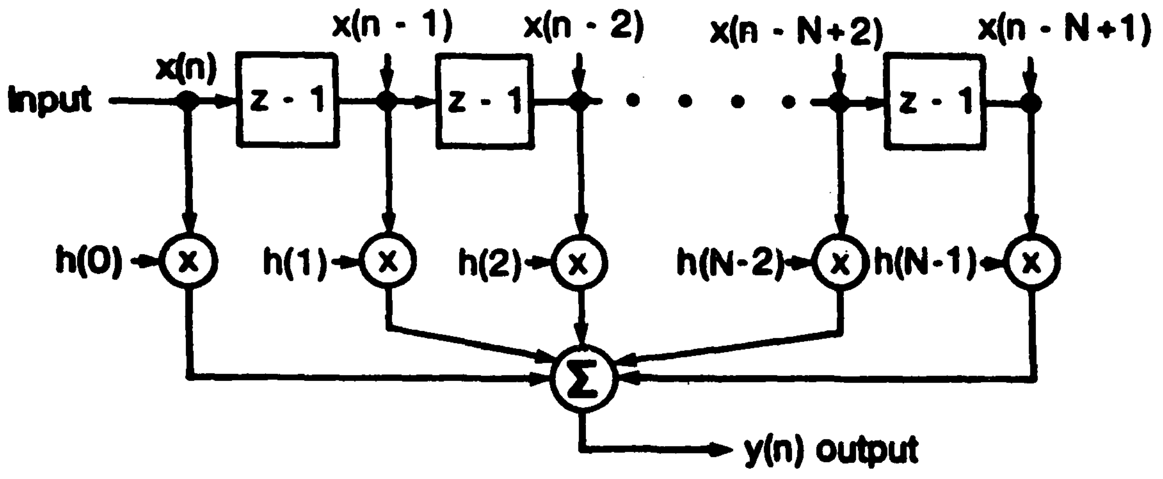


Figure 2.2

Direct Implementation of a Nonrecursive Digital Filter

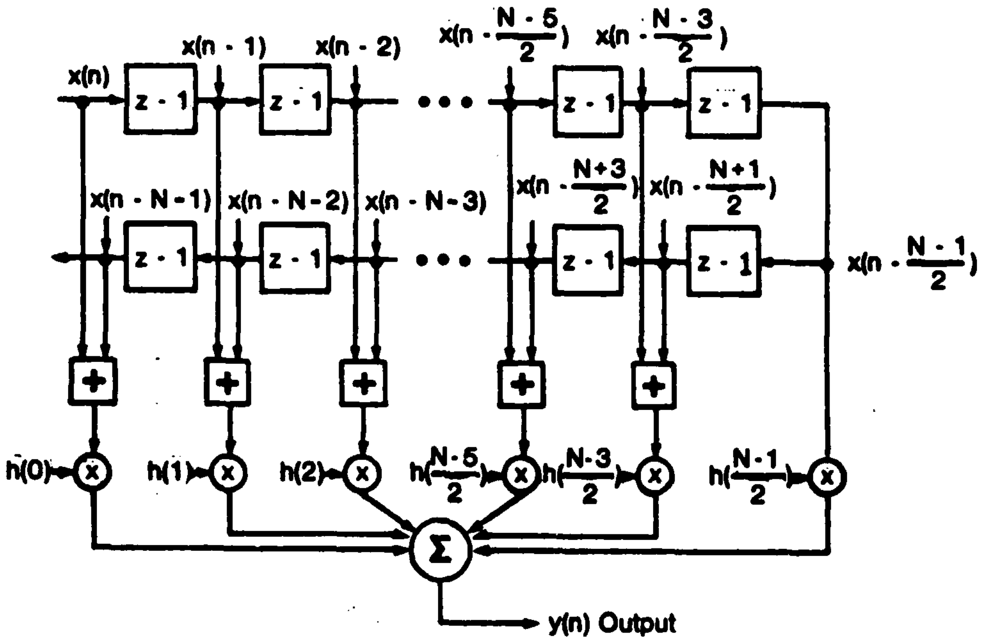


Figure 2.3

Linear Phase Finite Impulse Response Filter

2.2.3 Quantization Effects

A number of factors, or quantization effects, degrade the performance of a digital filter. Perhaps the largest contributor to error is the finite register length necessary for practical applications. For example, when two binary numbers are multiplied together, the product will require a number of bits equal to the sum of the number of digits in the two factors. This can quickly lead to registers that are prohibitively long. Usually, the product is rounded or truncated so as to consist of the same number of bits as the data word. This rounding or truncating, however necessary from a practical point of view, leads to a degradation in system performance.

Another source of error is in parameter quantization. Coefficients must be represented by a finite number of bits. This quantization can lead to some slight error in coefficient value. However, for reasonably long coefficient words, this effect becomes negligible.

The analog to digital conversion process measures the value of the analog signal at the sampling instant and assigns the amplitude to the nearest quantized value that can be represented digitally. This leads to some error between the true amplitude of the analog signal and the quantized level. This A/D conversion noise can lead to some error in the output. Again, however, with a reasonably precise A/D converter with a reasonably wide number of bits, this error is largely overshadowed by the errors involved with the necessity of finite length registers.

Selecting a register length for data and coefficients is based on compromising quantization effects with circuit size, complexity, and clock speed. The A/D conversion is a difficult task for long word lengths and fast clocks. Data and coefficient registers of 12 bits each were selected for this project considering these tradeoffs and the design objectives stated previously.

2.2.4 Coefficient Generation

The basic parameters of the magnitude versus frequency response for a bandpass filter are shown in Figure 2.4. Passband and stopband ripples are designated by $\sigma 1$ and $\sigma 2$, respectively. The passband is designated for frequencies in the region $f2 \leq f \leq f3$. Stopband regions are defined for $f \leq f1$ and for $f \geq f4$. Transition regions are given by $f1 \leq f \leq f2$ and $f3 \leq f \leq f4$. Normalized passband and transition widths can be defined in terms of the sampling frequency, fs, as follows:

$$\text{normalized passband} = BW = (f3 - f2)/fs$$
$$\text{normalized transition width} = (f2 - f1)/fs = (f4 - f3)/fs$$

The response of a fixed finite length digital filter at a given sampling frequency trades off optimal passband and transition widths and the amount of ripple present. For example, suppose a response that allows a 25–kHz passband with 25–kHz transition regions is desired for an 11th order filter with a fixed sampling frequency of 5 MHz. The selection of coefficients for this application will yield a response that has certain amount of passband and stopband ripple. The stop and ripple is directly related to the amount of out of band rejection for a filter. Now suppose the transition widths are changed to 125 kHz. The amount of ripple will be significantly less, but this improvement is achieved at the high cost of widening the transition regions. Changing the width of the passband region while maintaining the same transition widths will have a similar effect. Choosing the optimal combination of transition widths and allowable passband and stopband ripple, presupposing that the passband width is defined and unchangable, is a process that often

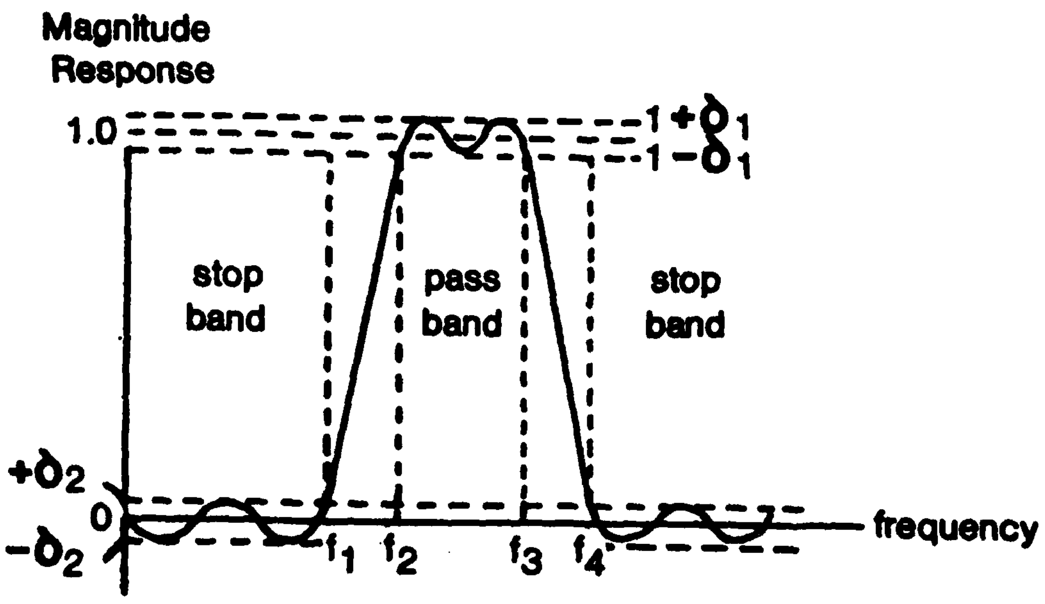


Figure 2.4

Magnitude vs. Frequency Response for a Digital Bandpass Filter requires many iterations and at times approaches an art. Luckily, programs exist that calculate normalized coefficients when given passband and stopband region definitions for filters of a given length.

Filter coefficients determine the exact response of a filtering system for a given filter length and sampling frequency. A program named FIRDESIGN [9] exists on the RICC VAX 785 that aids in coefficient calculation. The program is distributed by the IEEE Acoustics, Speech, and Signal Processing Society. The program uses the Remez exchange algorithm to calculate coefficient values for linear phase finite impulse response filters which are optimized with regard to minimizing the ripple in the pass and stop bands.

2.3 DESIGN APPROACH

2.3.1 Reproducible Slice Approach

The diagram shown in Fig. 2.3 can easily be broken up into several identical, regular portions, or "slices." Reproducible slices of the linear phase FIR filter are shown in Fig. 2.5. Sample (n–1) comes from the previous slice, while sample (n – N – i) returns from the next slice. The samples are added and the sum is weighted by a factor h(i) to produce a partial result p(i). Summing the partial results of each slice yields the filtered output y(n).

Three different types of slices can be defined: the first slice, the middle slice (s), and the last slice. The first and middle slices are identical to Fig. 2.5(a) except that the first slice has no need to pass on the returning sample, and the incoming sample is the digitized analog signal instead of a past data sample passed along by a previous slice. For all practical purposes, however, the first and middle slices perform the same function in that both process two samples.

The last slice, shown in Figure 2.5(b), is unique in that it processes only one sample and, rather than pass that sample on to the next slice, returns it to the previous slice. This accounts for the symmetry afforded by the linear phase approach. The component organization of the last slice can be identical to that of the first and middle slices, with the unique functions realized by appropriate wiring. This allows for a more regular layout and easier interconnection between slices.

2.3.2 Floorplan

Figure 2.6 shows a floorplan for a slice of the filter. Included in this slice is part of the final summation circuitry necessary to add all of the partial results p(i) to yield the filtered output y(n). The actual design of the chips uses data and coefficient word lengths of 12 bits each. However, the architecture presented hereafter is in no way constrained by these numbers. It is easily expanded for any lengths selected for data and coefficients words.

2.3.2.1 Addition of Data Samples

The first five rows of the floorplan shown in Figure 2.6 accomplish the addition of the data samples and convert the sum to a positive number if it is negative.

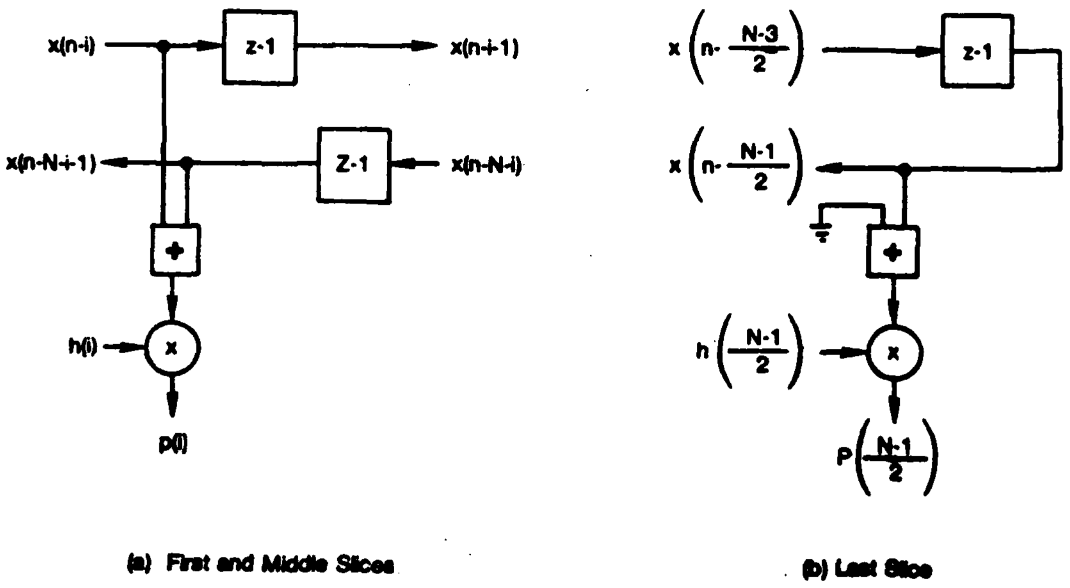

x(n-i)
z-1
x(n+1)
x(n-N+1)
Z-1
x(n-N-i)
+
h(i)
x
p(i)
(a) First and Middle Slices
$x\left(n-\frac{N-3}{2}\right)$
z-1
$x\left(n-\frac{N-1}{2}\right)$
+
$h\left(\frac{N-1}{2}\right)$
x
$P\left(\frac{N-1}{2}\right)$
(b) Last Slice

Figure 2.5

Reproducible Filter Slices

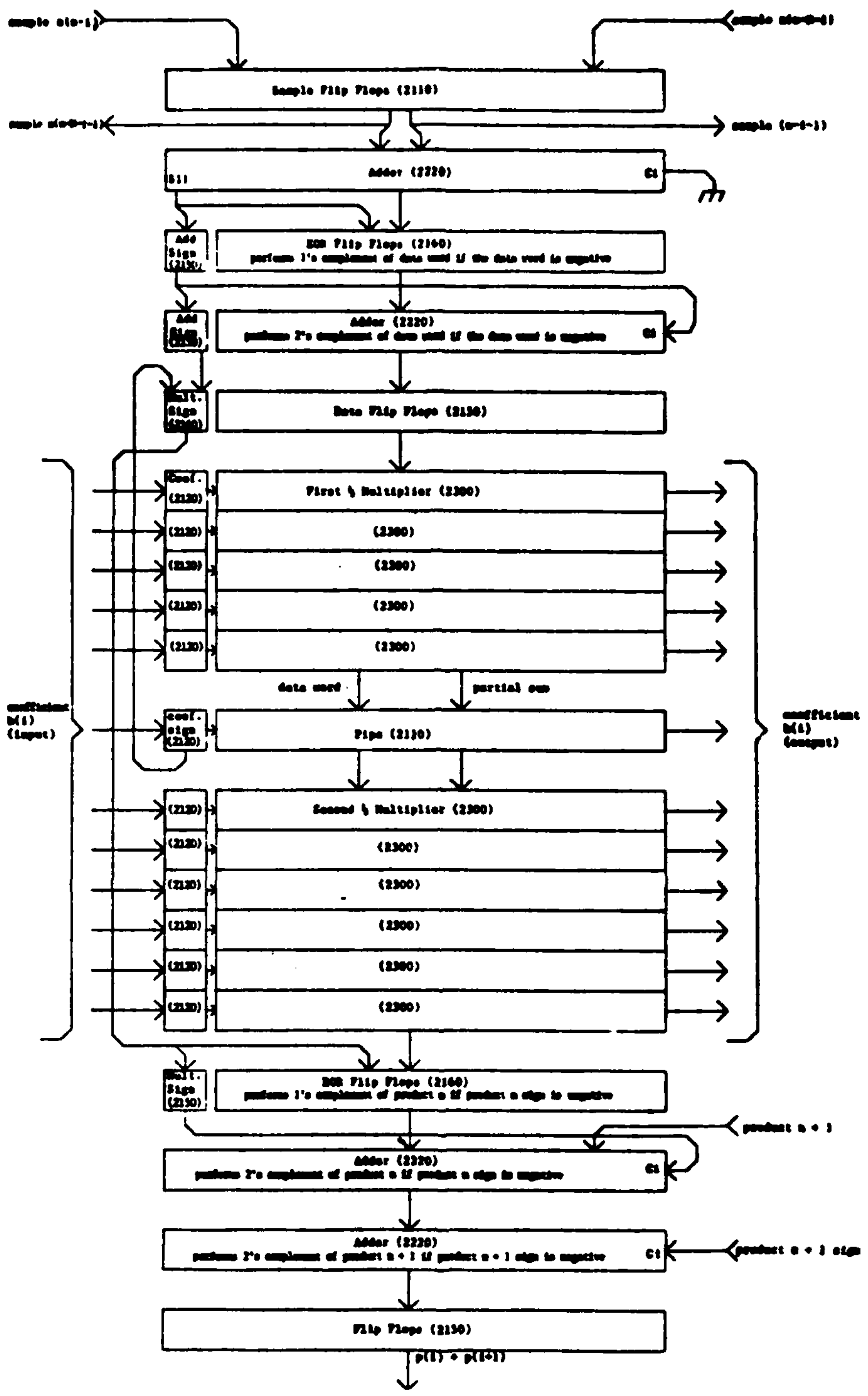


Figure 2.6

Floorplan for a Filter Slice

The first row in the floorplan contains dual flip flop cells to hold the samples. Data samples are stored in two's complement form. The samples are added by the second row of cells. No overflow detection is provided. Recovery from overflow is a difficult problem, and any recovery procedure will still yield results that are not consistent with results obtained during normal (i.e., no overflow) operation. Thus it was decided that if the filter output would be inaccurate anyway, no effort would be spent to minimize the inaccuracy. This is the usual approach taken towards overflow problems in digital filters. In any case, the overflow problem can be avoided by supplying an input signal that is sufficiently attenuated such that any two sample values, when added, will not cause overflow. This is easily accomplished in a digital filtering system by dividing down the analog input before supplying it to the A/D converter then amplifying the D/A output by the necessary amount once the filtering is accomplished.

The next three rows convert the sum to its two's complement positive representation if it was negative and preserve the original sign of the sum. This conversion is necessary because the parallel multiplication approach used works properly only for positive numbers. The third row consists of flip flops that perform an exclusive–or function on two inputs then store the result. This implements an efficient way to perform a one's complement operation. Flip flops were used for timing purposes in order to break up the combinational logic functions and speed up overall operation. The fifth row consists of an adder to complete the two's complement conversion process and a flip flop to store the original sign of the sum. The last row of the data sample addition section of the floorplan consists simply of a register used to store the data factor for the multiplier and an exclusive–or flip flop that uses the coefficient sign along with the original sign of the sum to determine the sign the product will carry.

The last slice of the filter utilizes the same component organization described above, but implements appropriate functions by appropriate wiring. The incoming data sample is clocked into both of the sample flip flops to accomplish the end–around data routing necessary in the last slice. The adder, instead of adding the outputs of both sets of flip flops, only accepts the output of one set of flip flops and adds this to zero to account for the wiring modification described above.

2.3.2.2 Coefficient Multiplication

The next 12 rows represent the coefficient register and multiplier. Coefficients are stored in sign–magnitude form. The coefficient registers of each slice are lined Q to D for efficient loading. When a coefficient load signal is activated, each coefficient flip flop sample passes the value at its input to its output. Thus a virtual common coefficient bus exists, with each bit of the coefficient register having the data on the bus at its input. The registers are then selectively clocked to store individual coefficient values. When the load signal is removed, the input–to–output link within each coefficient flip flop is broken, and each register ouputs its proper coefficient value to the multiplier.

The multiplier itself uses an (n–1) by (m–1) array of Guild cells to form the product, where n and m are the lengths of the coefficient and data words, respectively. Figure 2.7 shows the organization of the Guild parallel multiplier cell and the associated array structure that forms the multiplier. Coefficient bits are applied to each row of the array, and data bits to each column. The coefficient bits control the operation of the row to which they act as inputs. If the coefficient bit applied to a particular row is a zero, the partial sum input to that row is passed on unchanged, i.e., nothing is added to it. If the coefficient bit is a one, the data word is added to the partial sum in its proper position. Shifting of the data word to its proper position is accomplished by proper cell interconnection within the array. The product is output by the least significant cells of each row and by the final row in the array.

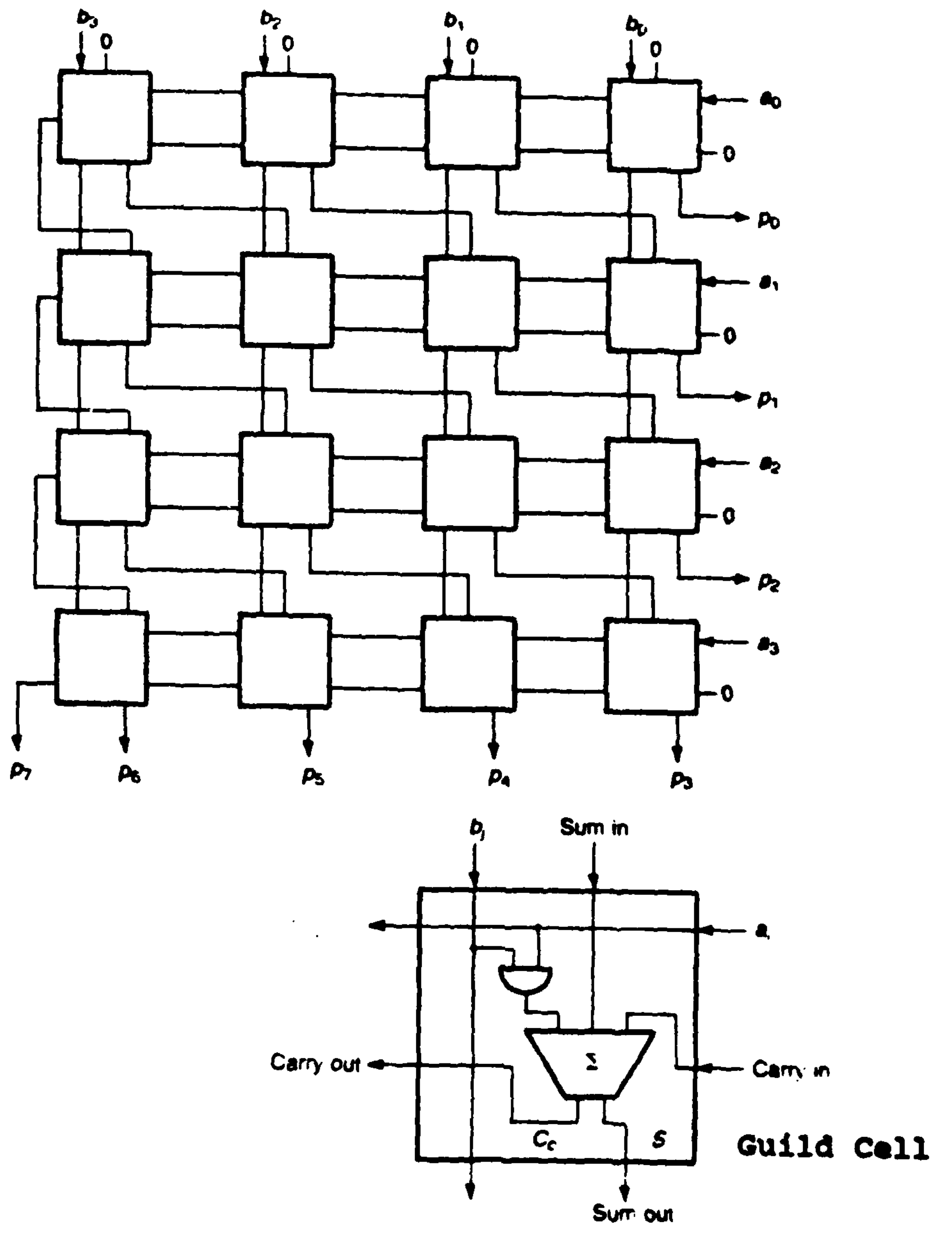


Figure 2.7

Multiplier Array Structure

The multiplication of an n bit number by an m bit number will require n+m bits to represent the full product. Due to finite register limitations, this product must be truncated or rounded to a lesser number of bits in digital filtering applications. The product is usually made to occupy the same number of bits as does the data word. The particular effects of truncating or rounding are for the most part indistinguishable; the main damage is done by cutting off the product bits of lesser significance. This design injects a high carry input value to the last row of the array, and limits the product to the most significant (m–1) bits. The effect of this is to round the product up to the most significant (m–1) bits.

The multiplier array described works only for positive numbers. Negative data word conversion is accomplished by the data sample addition section described in section 2.3.2.1. Coefficients are stored in sign–magnitude form. Since only positive numbers are being multiplied, the sign bits of the coefficient and data words contribute nothing to the result. Hence, an n by m array would be wasteful, since the product sign would always be positive. Therefore, the smaller array is used and the sign bits are not input to the multiplier. If the signs of the data and coefficient words indicate that the product is to be negative, the complementing process is carried out by the final summation circuitry described in section 3.2.3.

Propagation through the multiplier can be rather slow. In the worst case, a carry will need to propagate down the entire length of the first row in the array, then down an entire column. This means that the worst case propagation time will be approximately ((n–1) + (m–1)) * (carry propagation time for one cell). For this design, n and m are 12 bits each. This means that approximate propagation time through the array will be 22 carry propagation times.

The multiplication process can be speeded up by splitting the array into two halves and placing a pipeline between them. The data word and partial sum must be saved in the pipe. Now the worst case propagation time will be approximately ((n–1) + (m–1)/2) * (carry propagation time for one cell). This works out to 16 or 17 carry propagation times, depending on whether the first or second multiplier half is under consideration. This is approximately 3/4 the time required were the pipe not in place. The speed benefits provided by such a simple enhancement technique were deemed desirable and were therefore undertaken.

The coefficient register and multiplier array organization are identical for the first, middle, and last slices of a filter design.

2.3.2.3 Final Summation

The final four rows of the floorplan simultaneously complement the output of the multiplier if the product sign is negative and adds in the product of the next slice. Exclusive–or input flip flops are again used to perform efficient one's complementing based on the product sign. Two adders are necessary to complete the two's complementing process because of the possibility of both products being negative. The final row of the floorplan is a register used to store the sum of the products.

The final three rows of the floorplan are shared between every two slices. The outputs of every two slice combinations still need to be added together to form the final filtered output y(n). There are no necessary modifications to the last slice as long as the total number of slices is even. However, if there are an odd number of slices, one of the slices (presumably the last slice) will have to account for the extra summation circuitry present by tying the next–slice adder inputs to zero.

2.4 CELL DEFINITIONS AND SPICE SIMULATION RESULTS

A quick study of the floorplan just presented shows that six different custom cells are all that are needed for implementation. They are as follows:

1. A dual flip flop cell to hold samples and for the multiplier pipe (cell 2110);
2. A special multiplexed-output flip flop for use as a coefficient register element (cell 2120);
3. A standard flip flop for use in registers (cell 2150);
4. A flip flop cell with an exclusive-or gate at its input for efficient complement (cell 2160);
5. An adder cell (cell 2220);
6. A parallel array multiplier cell (cell 2300).

In addition to these customer cells, library cells for input and output functions are used.

All cells were laid out according to lambda-based scaleable design rules supplied by the MOSIS (MOS Implementation System) fabrication facilities based at the Informational Sciences Institute at the University of Southern California [4]. The lambda-based scaleable design rules allow a layout to be constructed using design rules based on a unit of length, lambda, which has no direct physical significance. This allows designs to be scaled down without having to create new layouts as technology improves and allows smaller device fabrication. This design assumes $\lambda = 1.5$ microns. If technology improves so that circuits for, say $\lambda = 1$ micron become common, the layout for this design should not need to be changed. Future chips will simply occupy less area.

For the most part, the nMOS design process prescribed by Mead and Conway [3] was followed; exceptions are noted in the following discussion. Layouts for all cells were made so as to appropriately stack with neighboring cells and allow ease of interconnect between cell rows.

SPICE [10] is a circuit simulator that produces the time response for a given network of resistors, capacitors, and semiconductor devices that are influenced by one or more inputs defined by the user. The Metheus/CV CAD system supports software tools that create network listings (netlists) from either schematics or physical layouts in a format acceptable as SPICE input. In this fashion, schematics can be simulated to provide an initial logic and timing verification of cell operation. Once verification is complete and a physical layout is constructed, a net list created from the layout is simulated. This netlist includes parasitic capacitances and therefore yields more accurate timing analysis. Parameter files for depletion and enhancement nMOS devices were constructed for use as device models using data provided by MOSIS for typical production runs.

In addition to the netlist to be simulated, SPICE allows additional circuit elements to be added to the network. Typically this includes additional resistors and capacitors used as load elements to more accurately model the output characteristics of a cell. SPICF will also allow simulation at different temperatures. To provide some semblance of worst case analysis, all cell simulations were carried out at 100 degrees C and with 0.5 pF load capacitance.

We should note that although SPICE is an excellent circuit simulator that provides accurate cell analysis, the scope of this simulator is small. It is intended for use when simulating small circuits, and is not appropriate for large circuit simulation. SPICE is a very computationally intensive program that requires a rather long amount of time to run. Typical simulations for the flip flops described below require about a half an hour of dedicated CPU time. Attempts to simulate circuits containing more than a few cells become prohibitively time consuming. In addition, the system runs out of swap space, i.e., runs out of memory, when attempts are made to simulate large circuits.

2.4.1 2110 Dual Flip Flop Cell

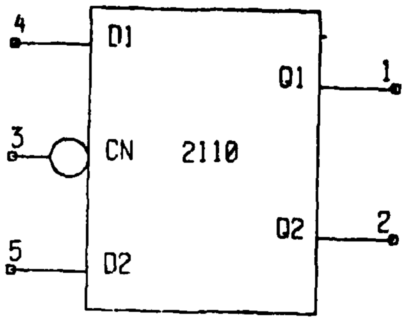
Figures 2.8a, b, and c show the symbol, schematic, and layout for the 2110 dual flip flop cell. Both flip flops are designed using the standard quasi-static feedback two-phase master-slave approach. This approach allows the old Q to circulate in the slave portion of the flip flop during one phase of the clock. During this time, the master accepts new data. When the two phases toggle, the value on the D input at the time the clock changed is circulated in the master portion of the flip flop and is passed through the slave portion to the output. When the phases toggle again, the new Q is held by the slave portion, and the master is once again able to accept new data. These flip flops were designed to change on the falling edge of the primary clock.

There are obviously setup and hold time requirements for such flip flops, but these requirements were largely ignored by this analysis. Such times are short, typically on the order of 3 to 5 ns, depending on the rise and fall times of the two clock phases and on their associated skew. Study of this design reveals that no serious race conditions occur. Setup times for all flip flops in this design will be at least (clock period) – (flip flop and logic propagation delay), which is at least on the order of hundreds of nanoseconds. The worst case hold time will be the propagation time through a flip flop. Typical propagation times for these flip flops is about 15 ns, which is well into the safety range for hold times.

There are two points of interest concerning this particular design. The first is that the outputs are driven by noninverting superbuffers rather than a simple inverter. The reason for doing this is obvious in that it greatly improves the drive capability of the flip flops without adding any real cost in terms of design time or silicon real estate. The superbuffer approach works by driving the depletion device of the second inverter with the input to the first inverter rather than by its own output. When trying to switch to a high, the gate of the depletion transistor of the second inverter goes high prior to the output going high. This yields a positive Vgs during the transition period of the output, thus turning the device on harder. This lowers the pullup resistance offered by the depletion load, allowing more current to charge output capacitances. At the same time, the output of the first inverter quickly turns off the enhancement mode device of the second inverter, since the gate of the enhancement device is the only load on the first inverter. A similar scenario occurs when trying to switch to a low, only this time Vgs for the enhancement device of the second inverter goes negative, thus increasing the pullup resistance. This allows the enhancement pulldown device of the second inverter to discharge a capacitive load more quickly. The end result is that rise times of the noninverting superbuffer are approximately half those that would be expected were the flip flop output simply inverted twice. Fall times are decreased somewhat also, but the improvement over the already excellent fall time characteristics of nMOS inverters is not so drastic. However, fall times will improve by a few percent.

The second interesting part is the fashion in which the device is clocked. A single clock input is inverted twice to produce the two phases necessary. This reproduction of the two phases occurs with each cell. This does not strictly produce a two-phase nonoverlapping clock as suggested by Mead and Conway [3] since, due to propagation delays, a short time will exist when both phases of the clock are active (high). This overlap can be neglected if the design meets the following criteria:

- The loads on each of the clock lines must be approximately equal so that the rise or fall time of one of the clocks is not slowed up to the point that it extends the overlap period to a critical point; and



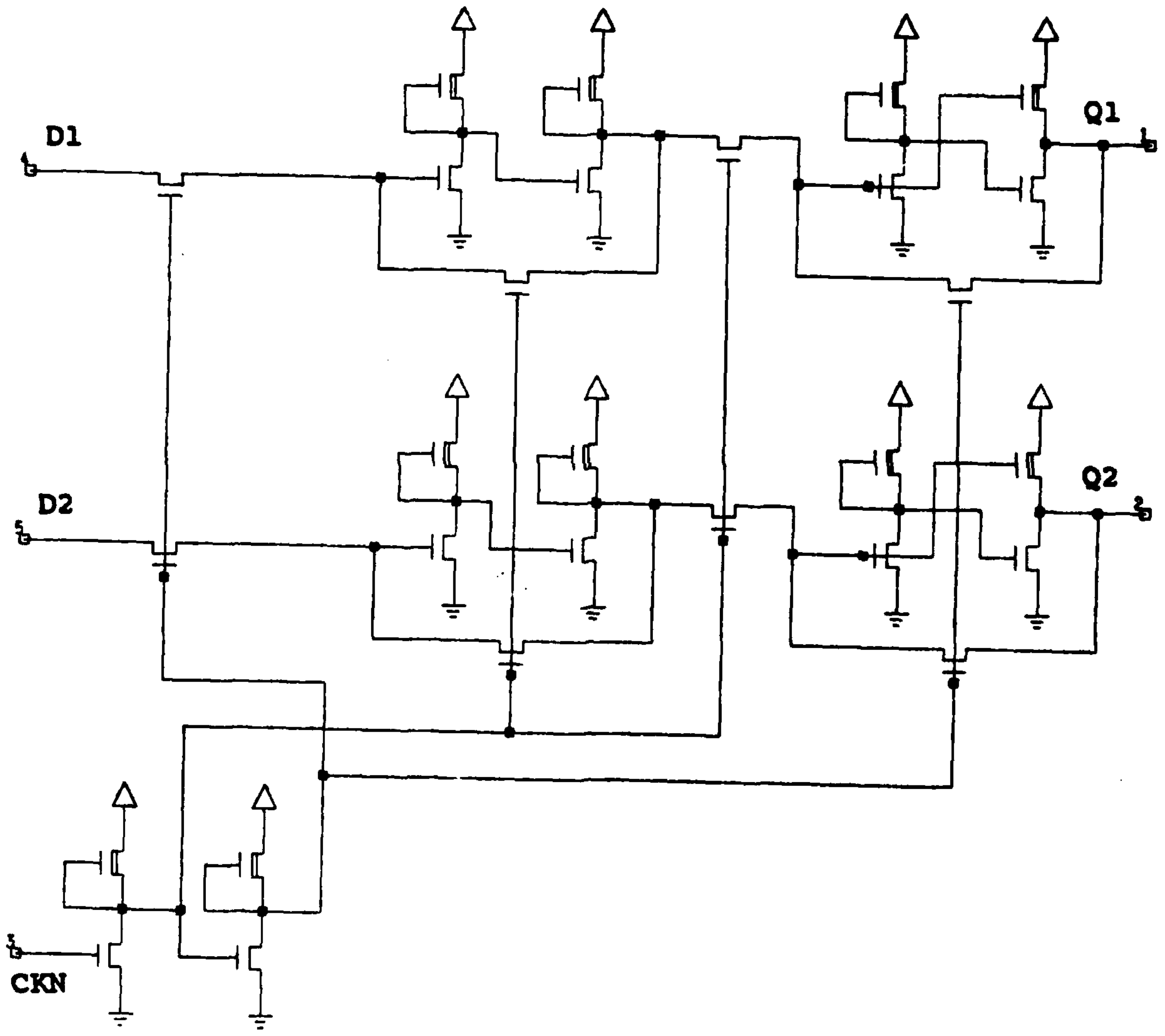


Figure 2.8 (a), (b)

(a) 2110 Dual Flip Flop Symbol
(b) 2110 Dual Flip Flop Schematic

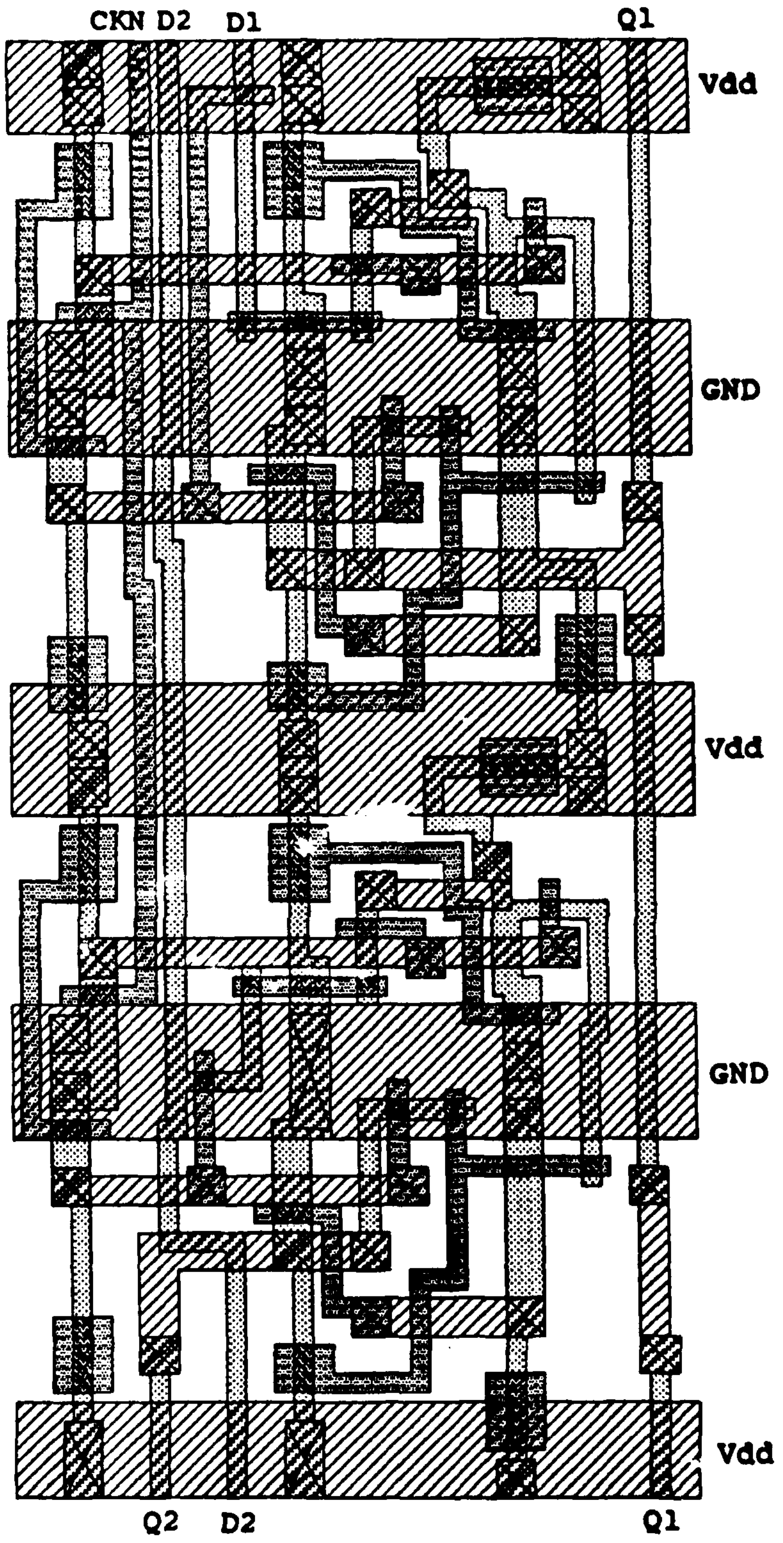


Figure 2.8 (c)

(c) 2110 Dual Flip Flop Layout

The load on the clock lines must be sufficiently small so that clock rise and fall times can be sufficiently fast so as to constrain the overlap to a sufficiently small period of time.

These criteria were verified by monitoring critical nodes within the flip flops during SPICE simulation runs.

The clock scheme described above has other advantages as well from the chip timing and interconnection point of view. First, only one clock needs to be routed around the chip. This eases wiring, reduces skew problems throughout the chip, and eliminates skew problems between the two clock phases, since theses phases are reproduced locally inside each cell. Also, because of this local reproduction, the system wide clock can have slower rise and fall times without degrading system performance.

The physical layout for this cell is contained in a 71λ by 154λ area. Power busses stack from side to side. Inputs and outputs emerge from the top and bottom of the cell according to necessity in order to allow easy interconnect with neighboring cell rows. SPICE simulations of the layout give worst case propagation delays of 15 ns (clock to Q) and rise and fall times of 39 ns and 6 ns, respectively, for an output load of 0.5 pF and at 100 degrees C. The SPICE output for one of the two outputs of this cell is shown in Figure 2.9. The graph for the other output is identical.

SPICE will also simulate the current drawn from a voltage source. Figure 2.10 shows the dynamic current drawn from Vdd for cell 2110. By monitoring the peak current values and making the worst case assumption that every cell in a row might hit the peak simultaneously, the minimum width for power lines through the cells can be calculated. MOSIS guarantees minimum sheet current carrying capability of 1 mA per micron width. By calculating the maximum current a row may draw, the appropriate width of power lines can be determined by using

(max current mA) / (1 mA/micron) * (0.667 lambda/micron),

assuming that $\lambda = 1.5$ microns. Similar analysis was carried out for the other custom cells used in this design and for the wide power busses that supply Vdd and ground to the cell rows.

2.4.2 2120 Multiplexed Output Flip Flop Cell

The symbol, schematic, and layout for the 2120 multiplexed output flip flop cell are shown in figure 2.11. The flip flop itself is identical to those of cell 2110, with the same output superbuffering and clock generation scheme. The unique feature of this cell is the multiplexer on the output that selects either Q or D, depending on the state of the LOAD input. The multiplexer itself is constructed simply of pass transistors, which degrades the output of this cell. However, this poses no serious problem since coefficient loading can be a relatively long process, and since the coefficients, once loaded, don't change for a given filtering application. The output of the multiplexer is pulled up by a very weak depletion mode transistor. This was done to insure that the output would rise the full range of Vdd so that other cells which accept this output as input would not need to consider inverter ratio requirements. Simulations were carried out to insure that both highs and lows could be propagated down the entire length of the bus when the virtual coefficient bus is in existence. The configuration described has no difficulty passing on a high value, but problems can develop when propagating a low if the output pullup devices are too strong.

This cell was constructed in a 71λ by 103λ area. Power busses are stacked from side to side. Inputs and outputs are brought out to the top and bottom of the cell for ease of interconnect. Clock and load inputs stack top to bottom to allow easy register

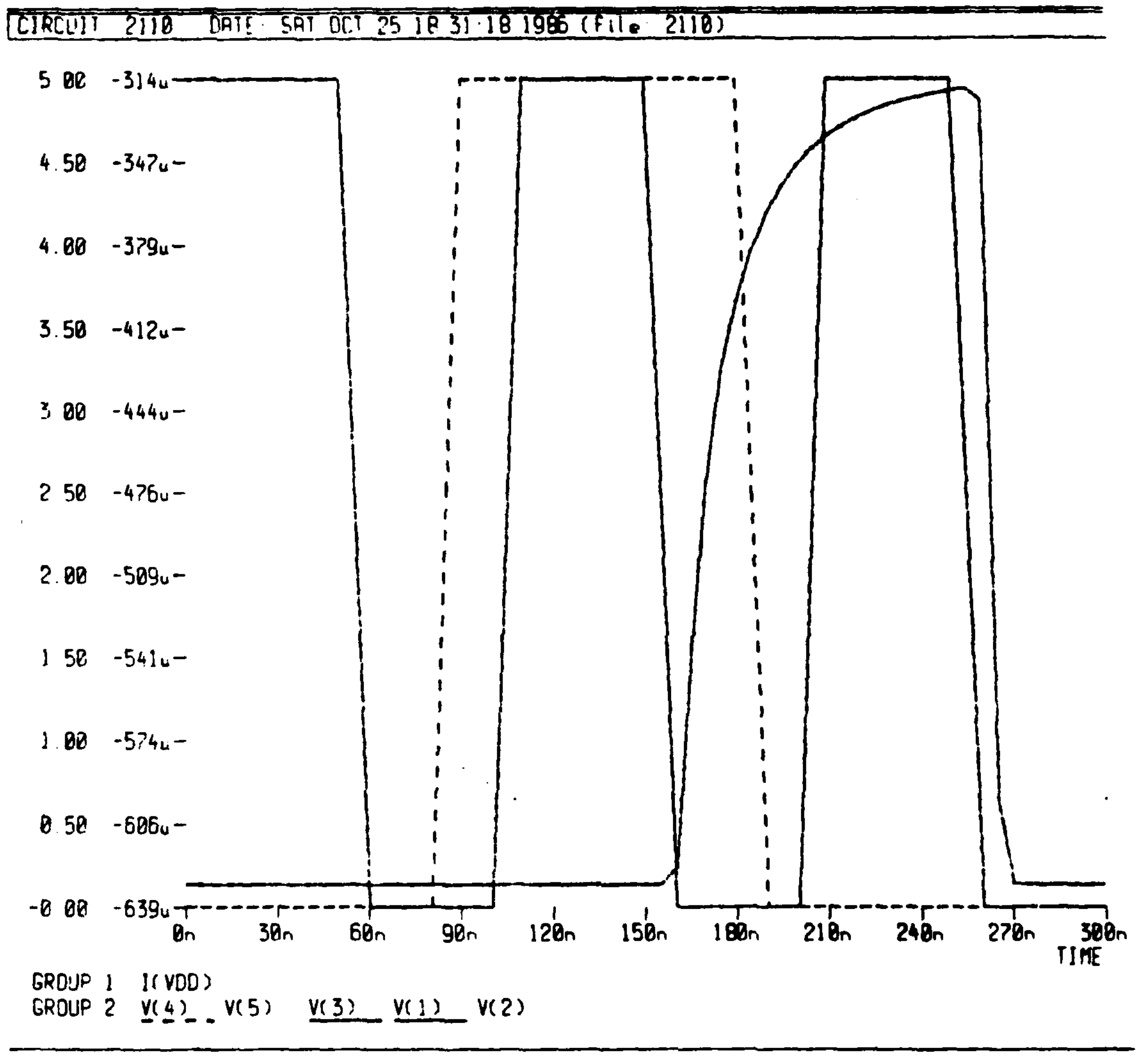


Figure 2.9

2110 Dual Flip Flop SPICE Output

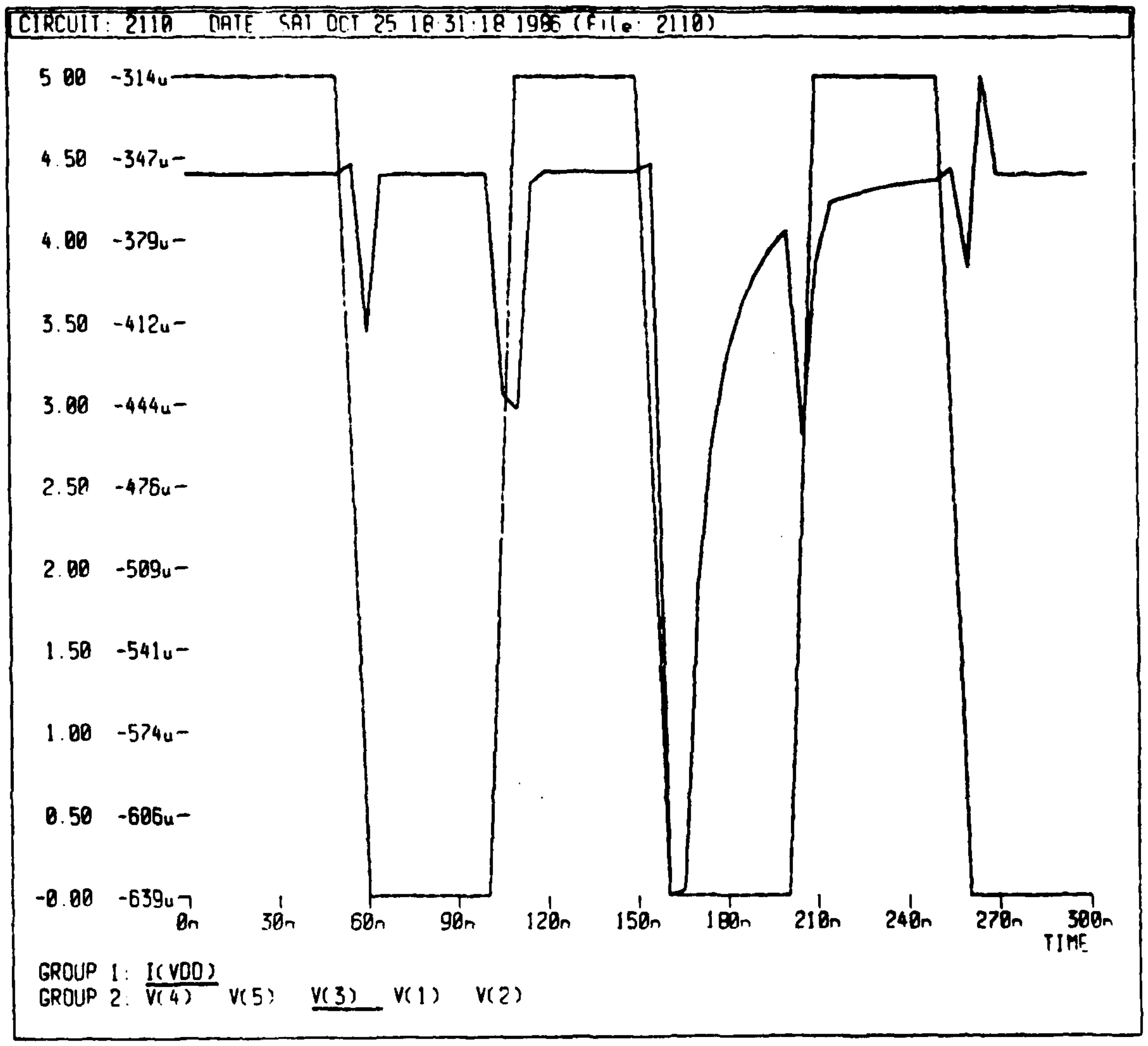
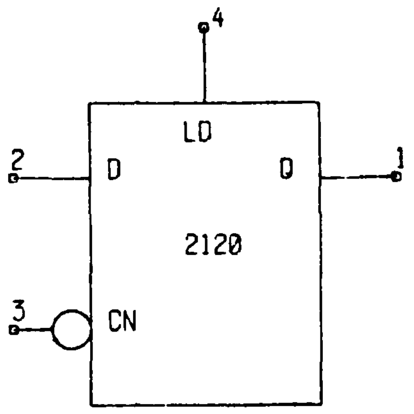


Figure 2.10

2110 Dual Flip Flop Daynamic Current Draw



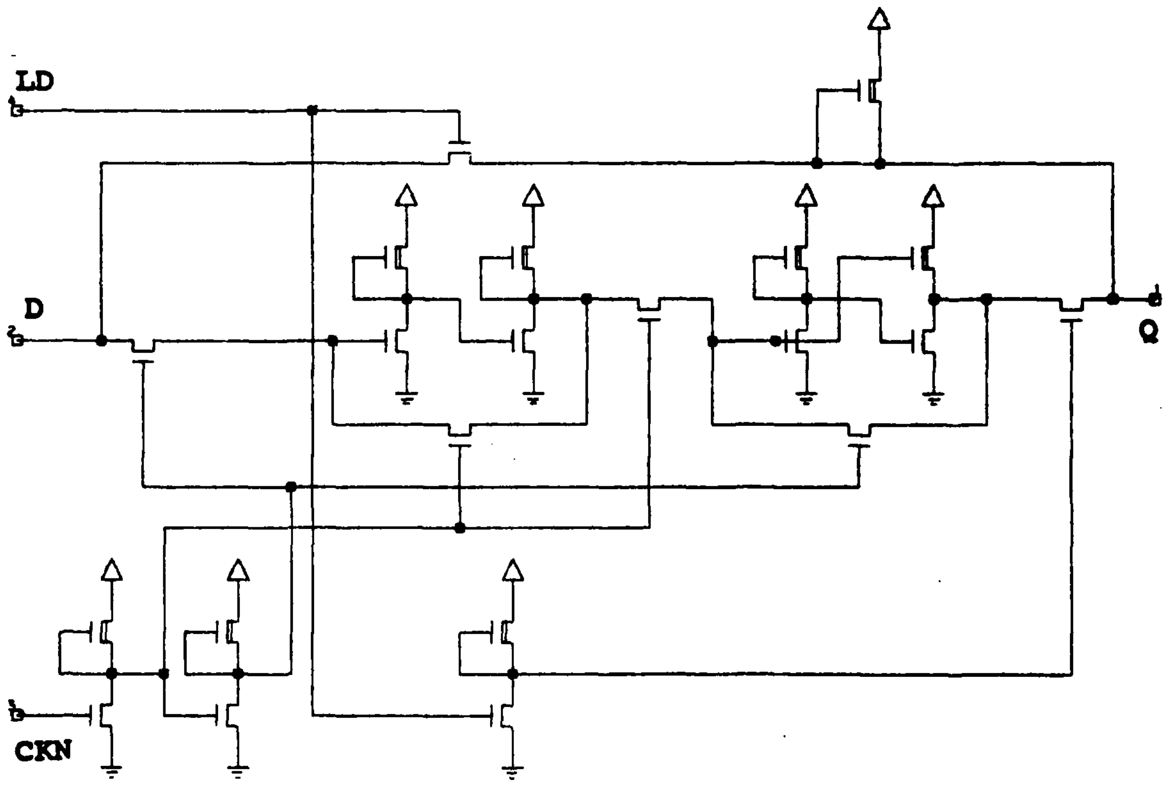


Figure 2.11 (a), (b)

(a) 2120 Multiplexed Output Flip Flop Symbol
(b) 2120 Multiplexed Output Flip Flop Schematic

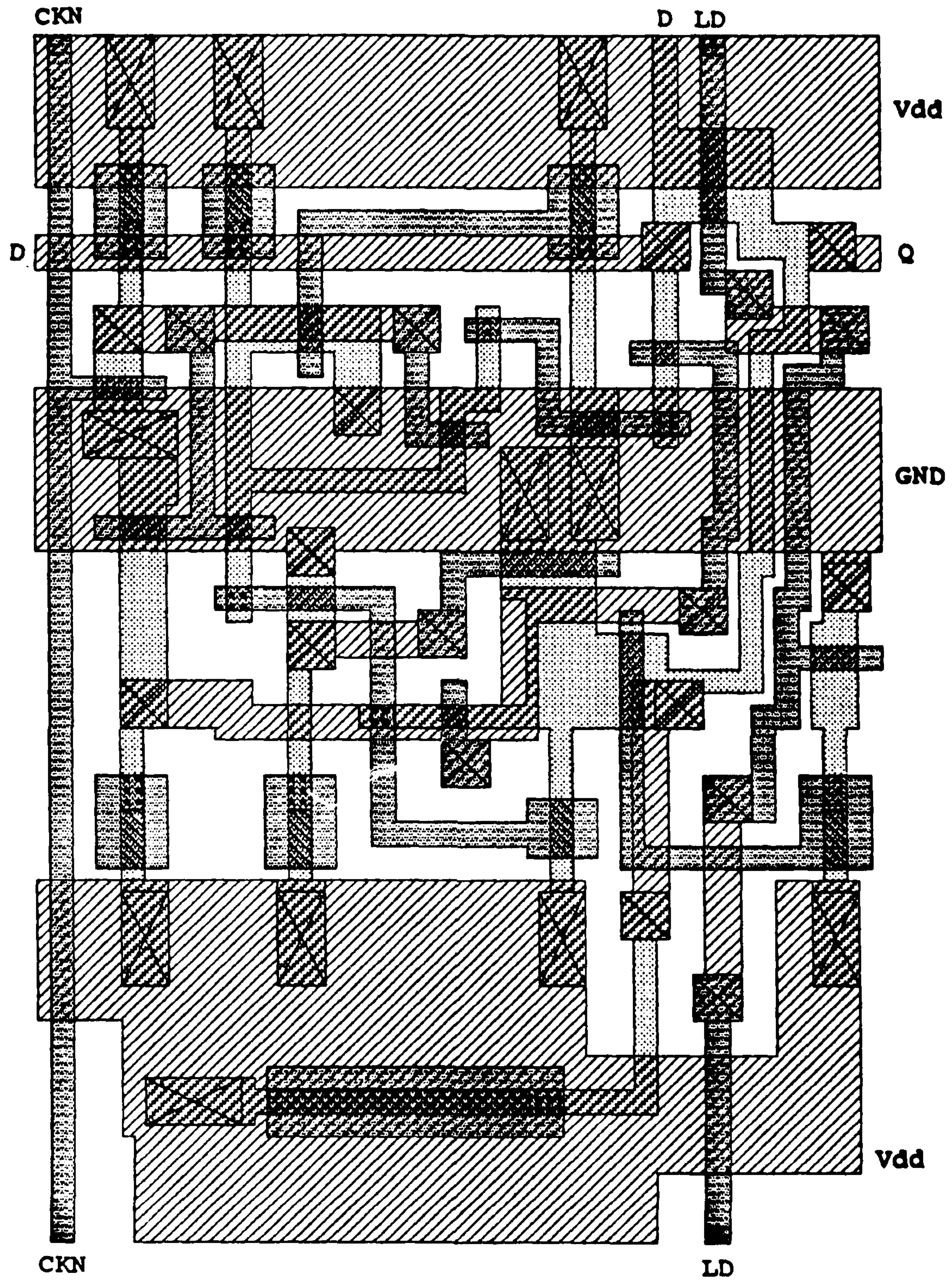


Figure 2.11 (c)

(c) Multiplexed Output Flip Flop Layout

interconnection. In addition, D and Q stack side to side to allow easy formation of the virtual coefficient load bus described earlier. The Q output also stacks with the coefficient input to the multiplier cells, since these cells are adjacent in the layout. SPICE simulations of the layout give a worst case propagation delay of 14 ns (clock to Q) and rise and fall times of 30 ns and 7.5 ns, respectively, for an output load of 0.5 pF and at 100 degrees C. Figure 2.12 shows graphical SPICE output for this cell.

2.4.3 2150 Standard Flip Flop Cell

Figure 2.13 shows the symbol, schematic, and physical layout for the 2150 standard D type flip flop cell. This cell is a single flip flop version of the 2110 dual flip flop cell described earlier and is identical in configuration. It employs the same output superbuffering and clock generation. The physical layout for the 2150 was constructed in a 71λ by 82λ area. SPICE simulations yielded a worst case propagation delay of 14 ns (clock to Q), with rise and fall times of 39 ns and 6 ns, respectively, for an output load of 0.5 pF and at 100 degrees C. Figure 2.14 shows the graphical SPICE output for the simulation of the timing characteristics of this cell.

2.4.4 2160 Exclusive–OR Input Flip Flop Cell

Figure 2.15 shows the symbol, schematic, and layout for the 2160 exclusive–OR input flip flop cell. This cell accepts two inputs and stores their exclusive–ORed output in the flip flop. Aside from the obvious difference in input characteristics, this device is different from the other flip flops in that the output is not driven by a superbuffer. This was done to facilitate the preservation of logic levels. The (exclusive) logic at the input actually implements an exclusive–NOR function the quick and dirty way using pass transistors. Figure 2.16 shows the schematic for this implementation. The design uses only three transistors, is easy to lay out, consumes very little silicon area, and works for this application. Drawbacks include poor driving capability, but this can be ignored in this case since the output does not drive any logic outside the cell and only drives one gate through a pass transistor inside the cell.

The value stored in the flip flop is the exclusive–NORed output of the two inputs. In order to restore proper logic levels, the Q' output is brought outside the cell. Superbuffering is not necessary since that the output load is sufficiently small for all instances where this cell is used.

The physical layout for this cell is contained on a 71λ by 82λ area. Power busses stack from side to side. Inputs are accepted at the top of the cell, and the Q output is available at both the top and bottom of the cell. SPICE simulation gives a worst case propagation delay (clock to Q) of 20 ns and rise and fall times of 60 ns and 6 ns, respectively, for an output load of 0.5 pF and at 100 degrees C. Figure 2.17 shows the simulation output for this cell.

2.4.5 2220 Adder Cell

The symbol, schematic, and physical layout for the 2220 adder cell are found in Figures 2.18a, b, and c. The adder employs a distributed gate approach to implement the sum and ripple carry functions. The basis of this approach is to combine NAND and NOR functions into a gate with a common pullup to implement and AND–OR–INVERT function. There

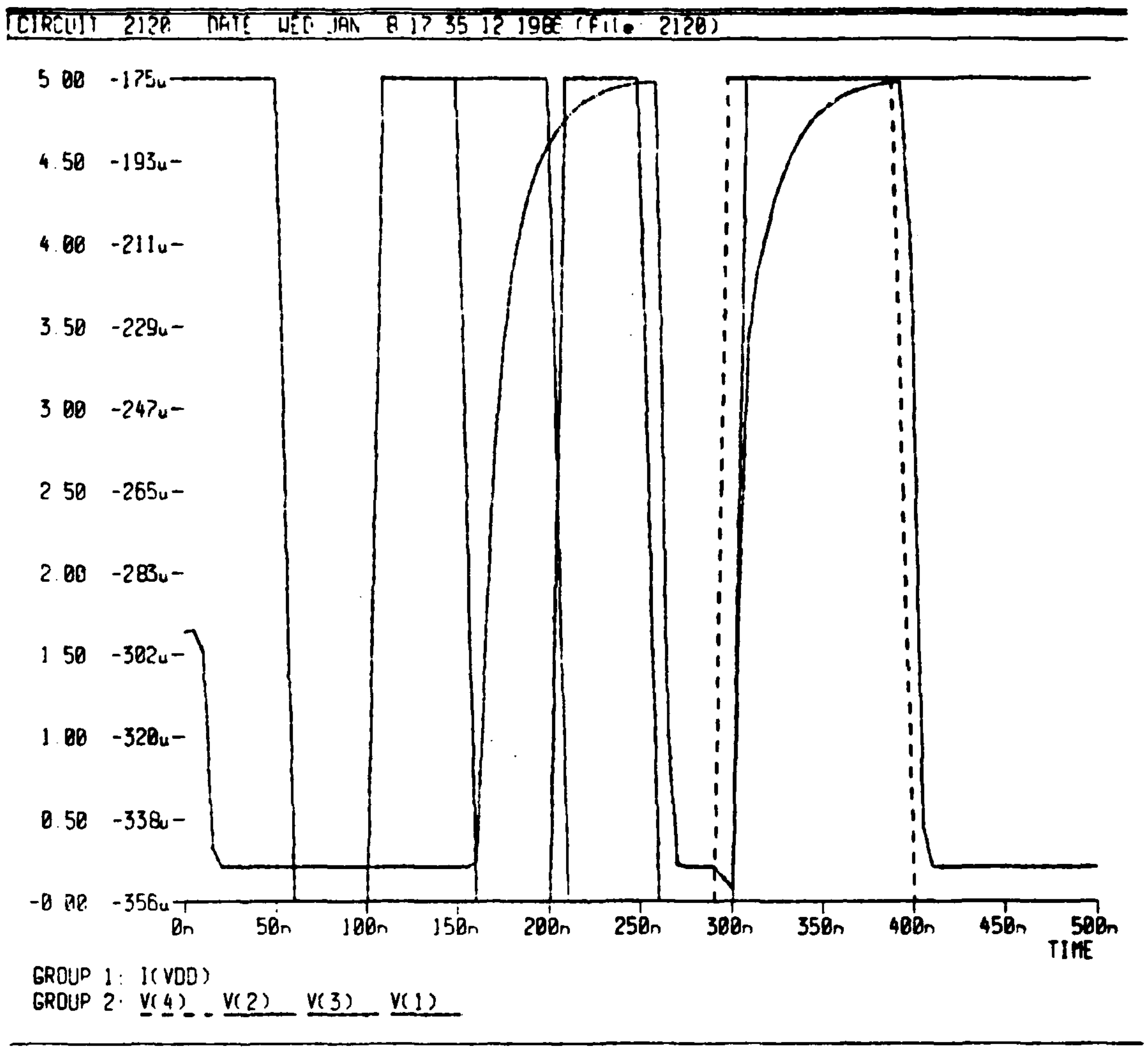
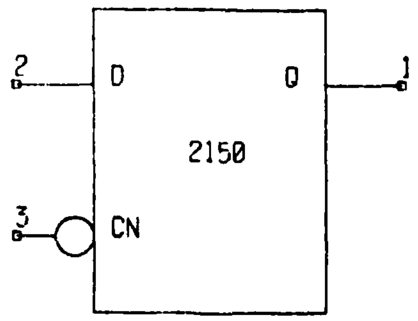


Figure 2.12

2120 Multiplexed Output Flip Flop SPICE Output



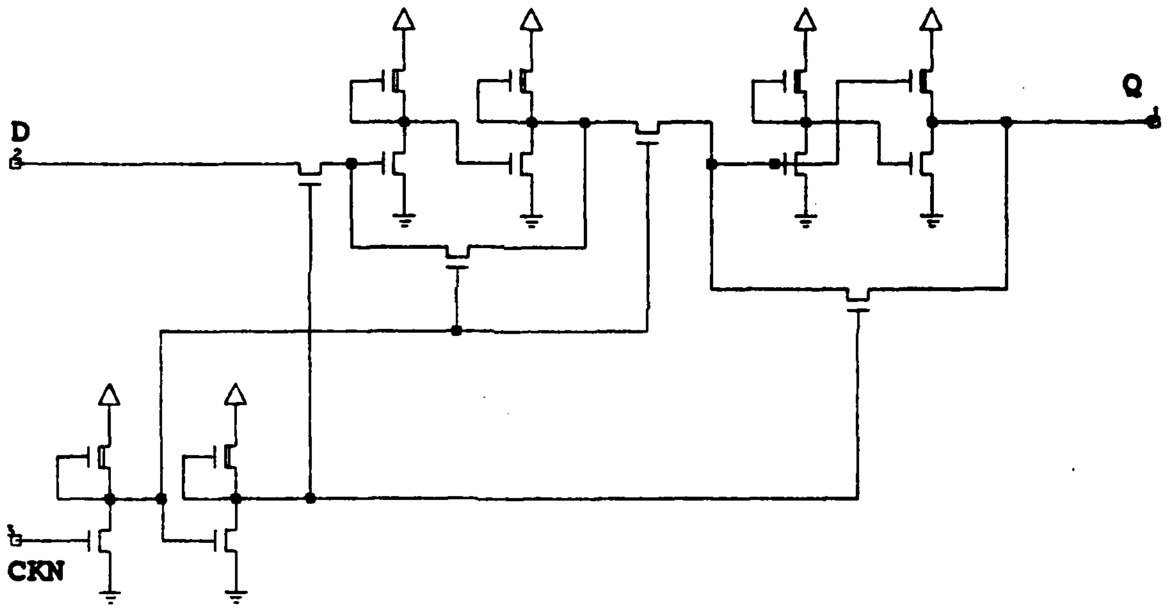


Figure 2.13 (a), (b)

(a) 2150 Flip Flop Symbol
(b) 2150 Flip Flop Schematic

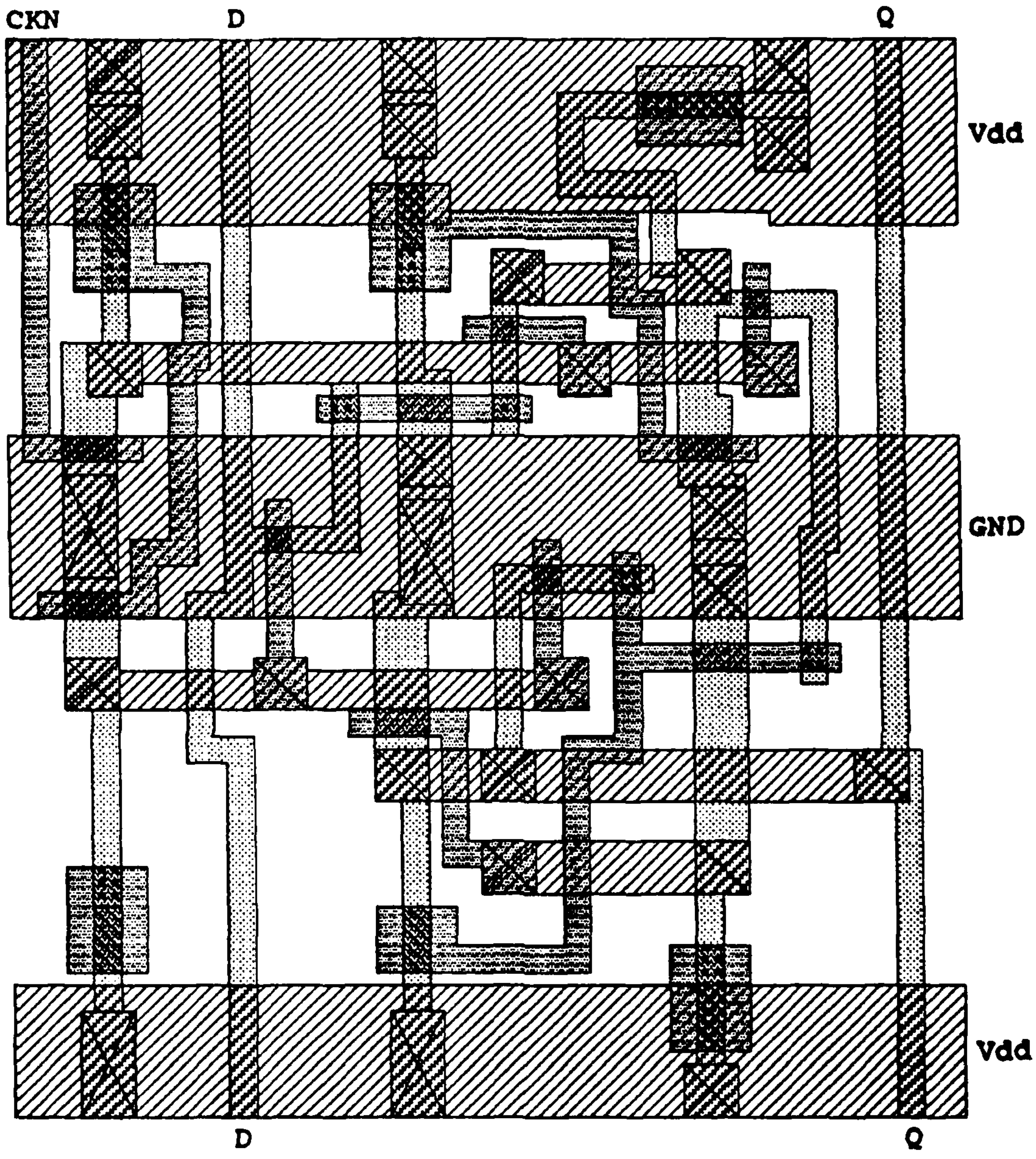


Figure 2.13 (c)

(c) 2150 Flip Flop Layout

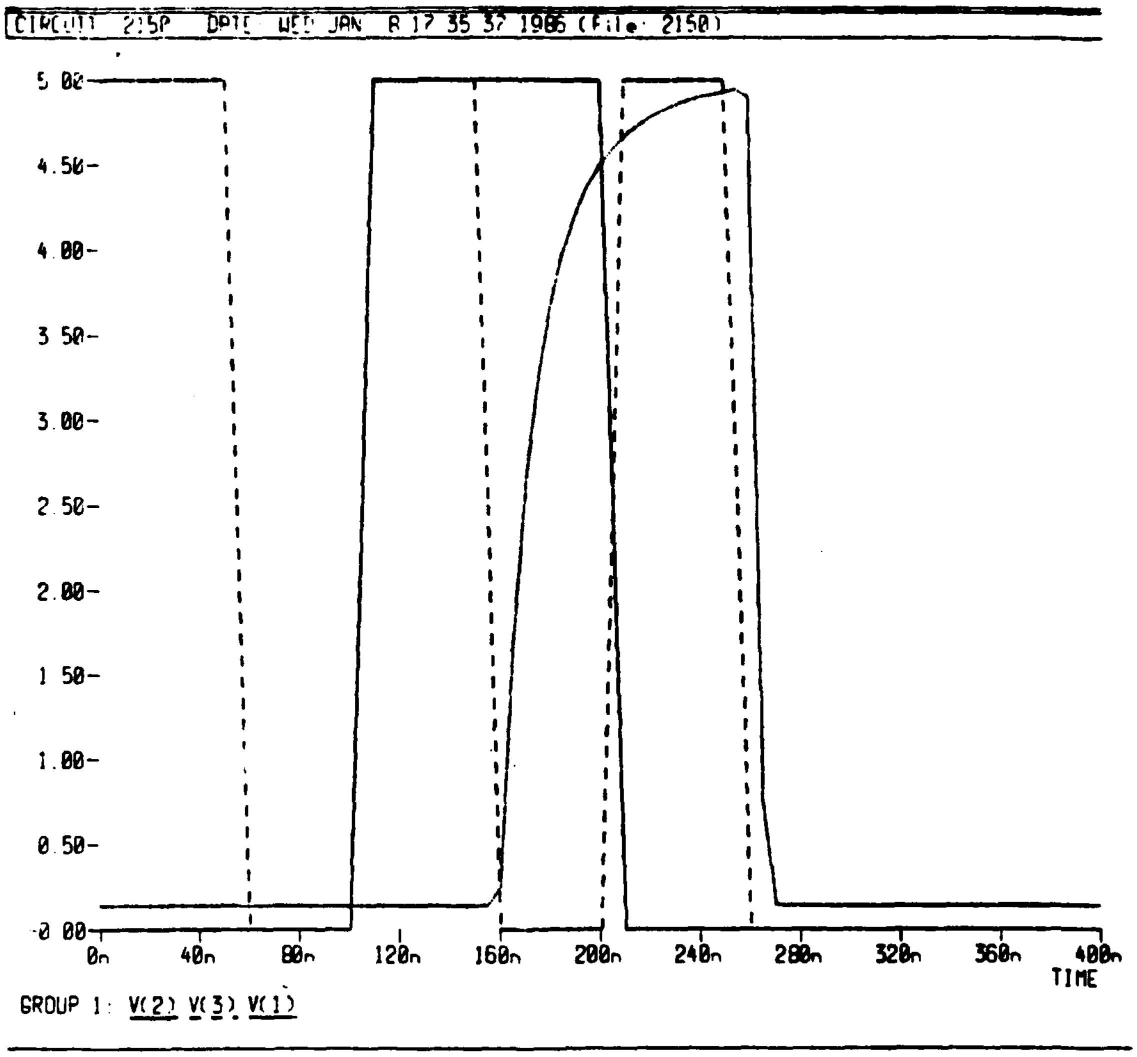
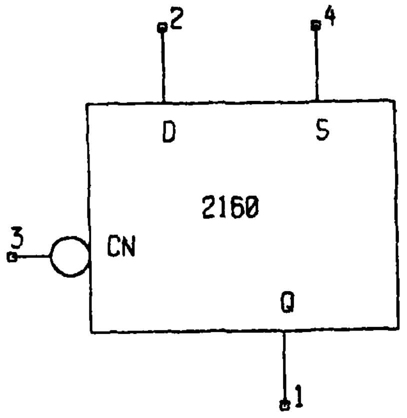


Figure 2.14

2150 Flip Flop SPICE Output



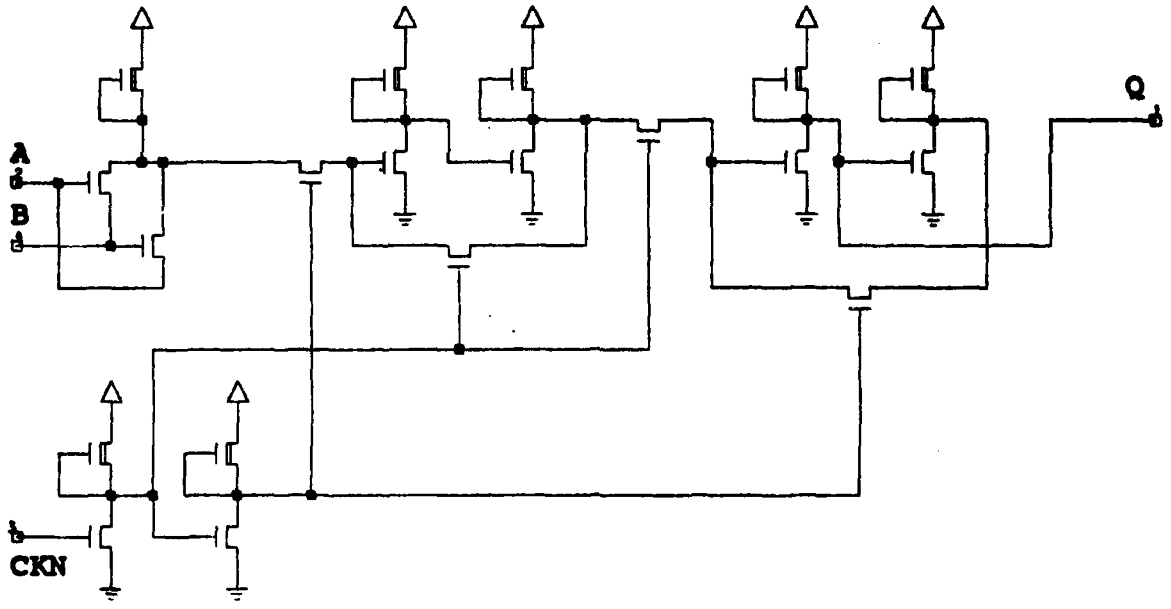


Figure 2.15 (a), (b)

(a) 2160 XOR Input Flip Flop Symbol
(b) 2160 XOR Input Flip Flop Schematic

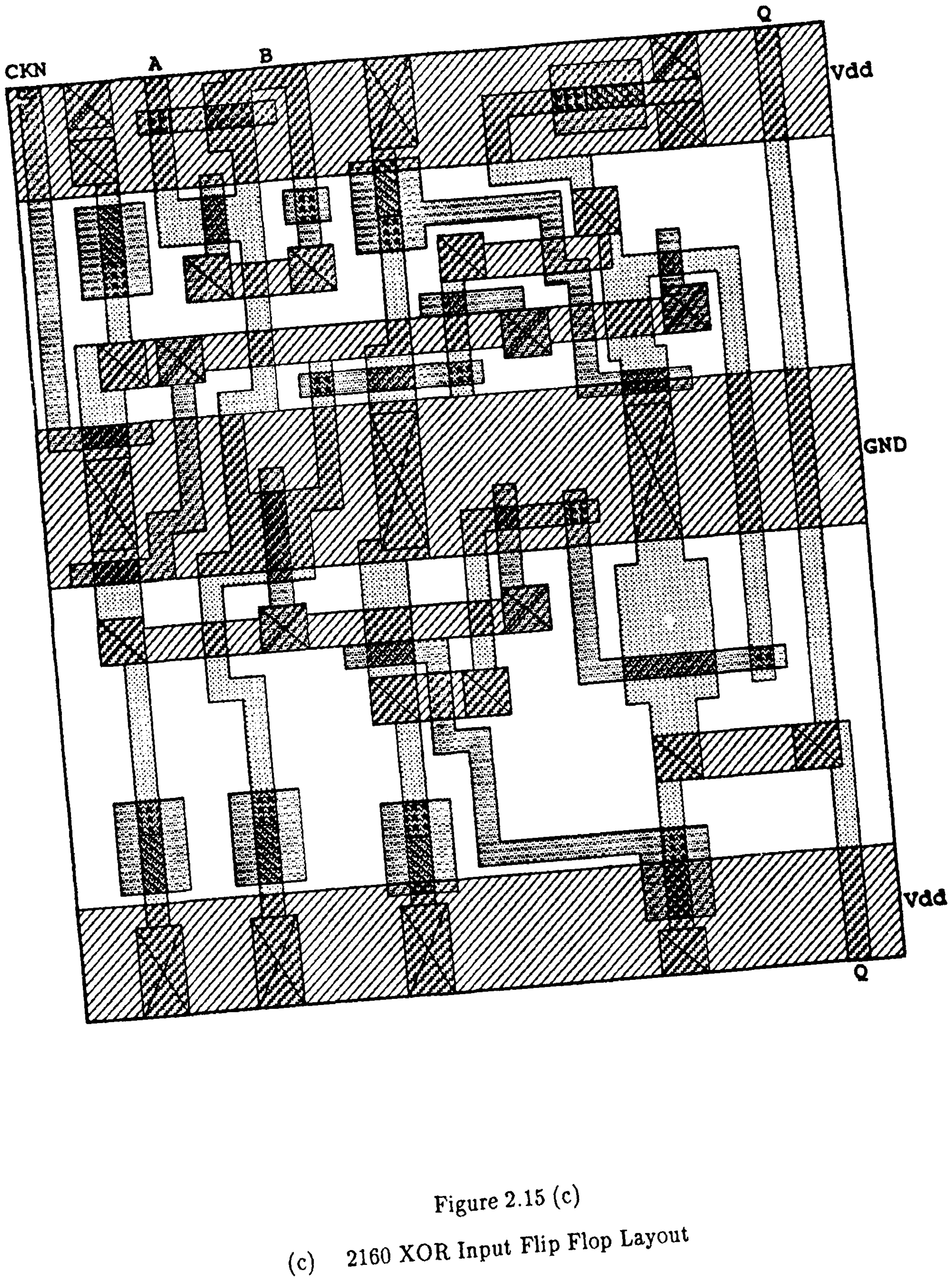


Figure 2.15 (c)

(c) 2160 XOR Input Flip Flop Layout

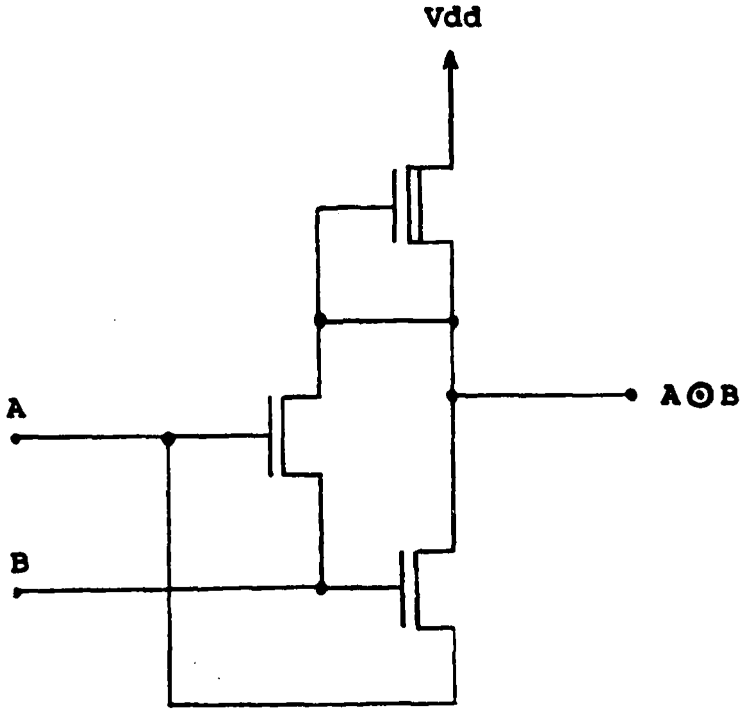


Figure 2.16

Efficient XNOR Implementation

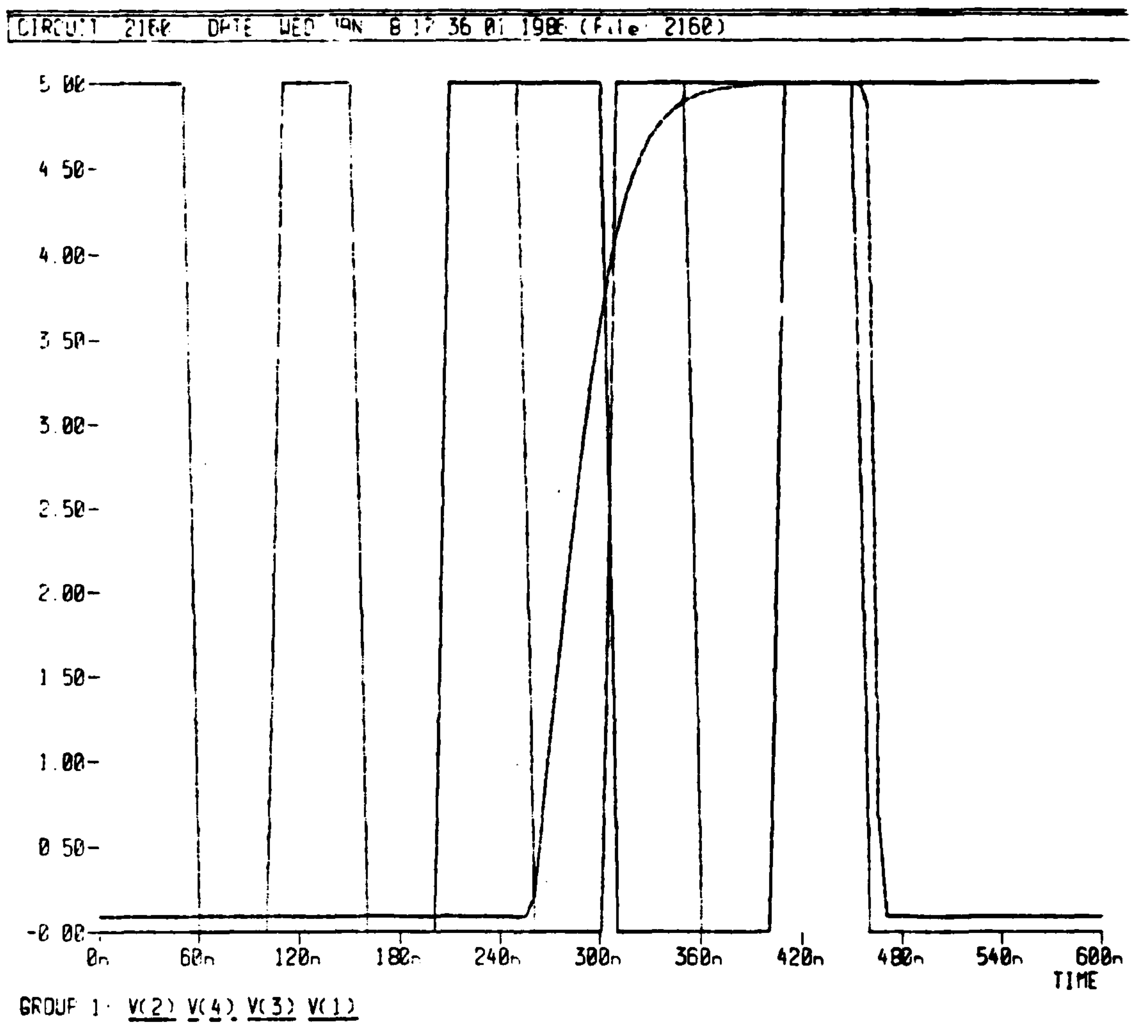
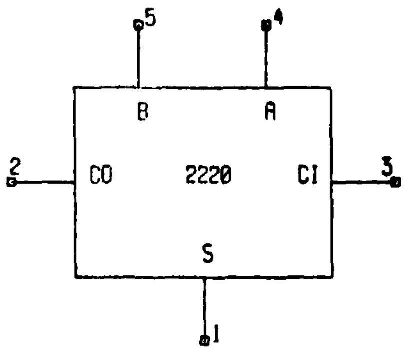


Figure 2.17

2160 XOR Input Flip Flop SPICE Output



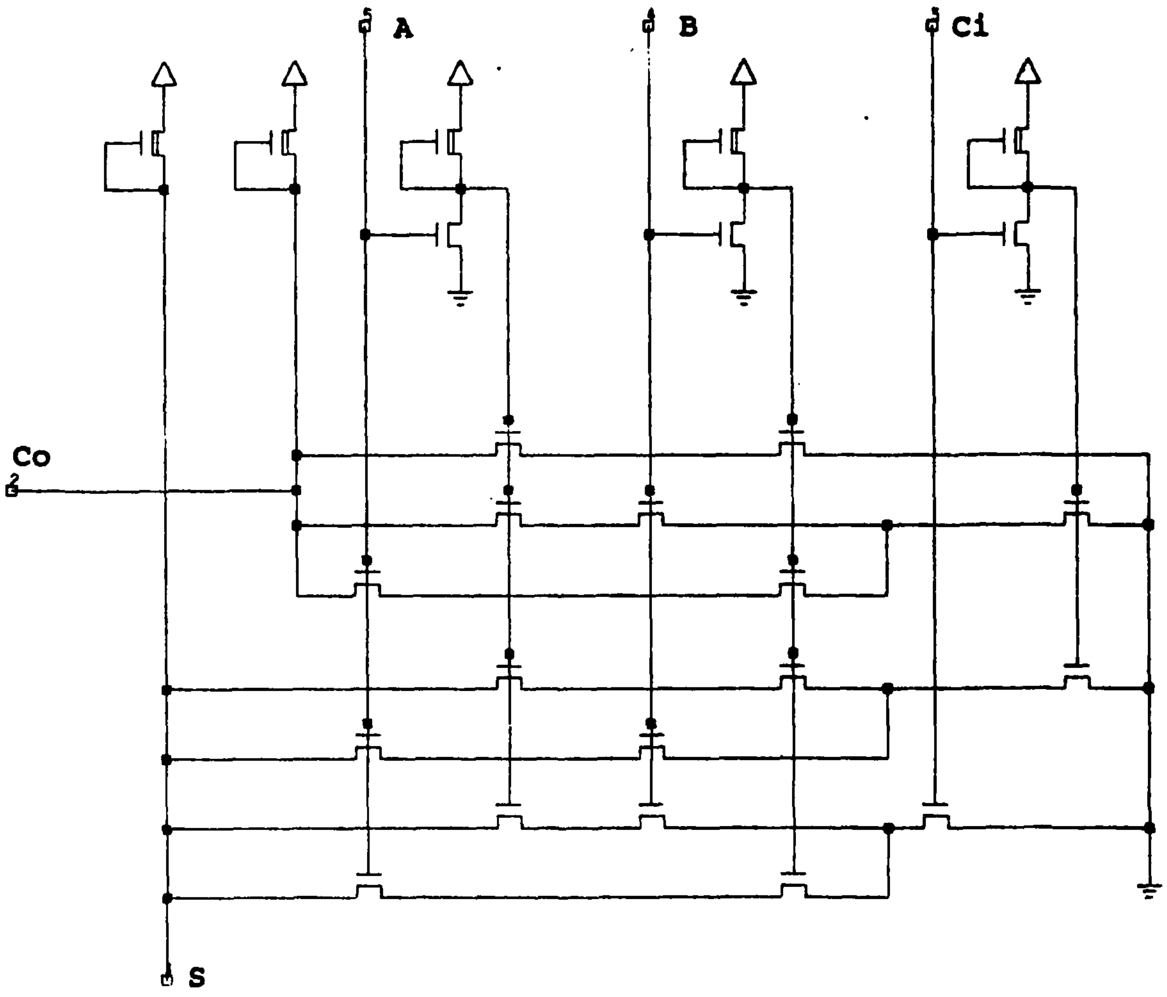


Figure 2.18 (a), (b)

(a) 2220 Adder Symbol
(b) 2220 Adder Schematic

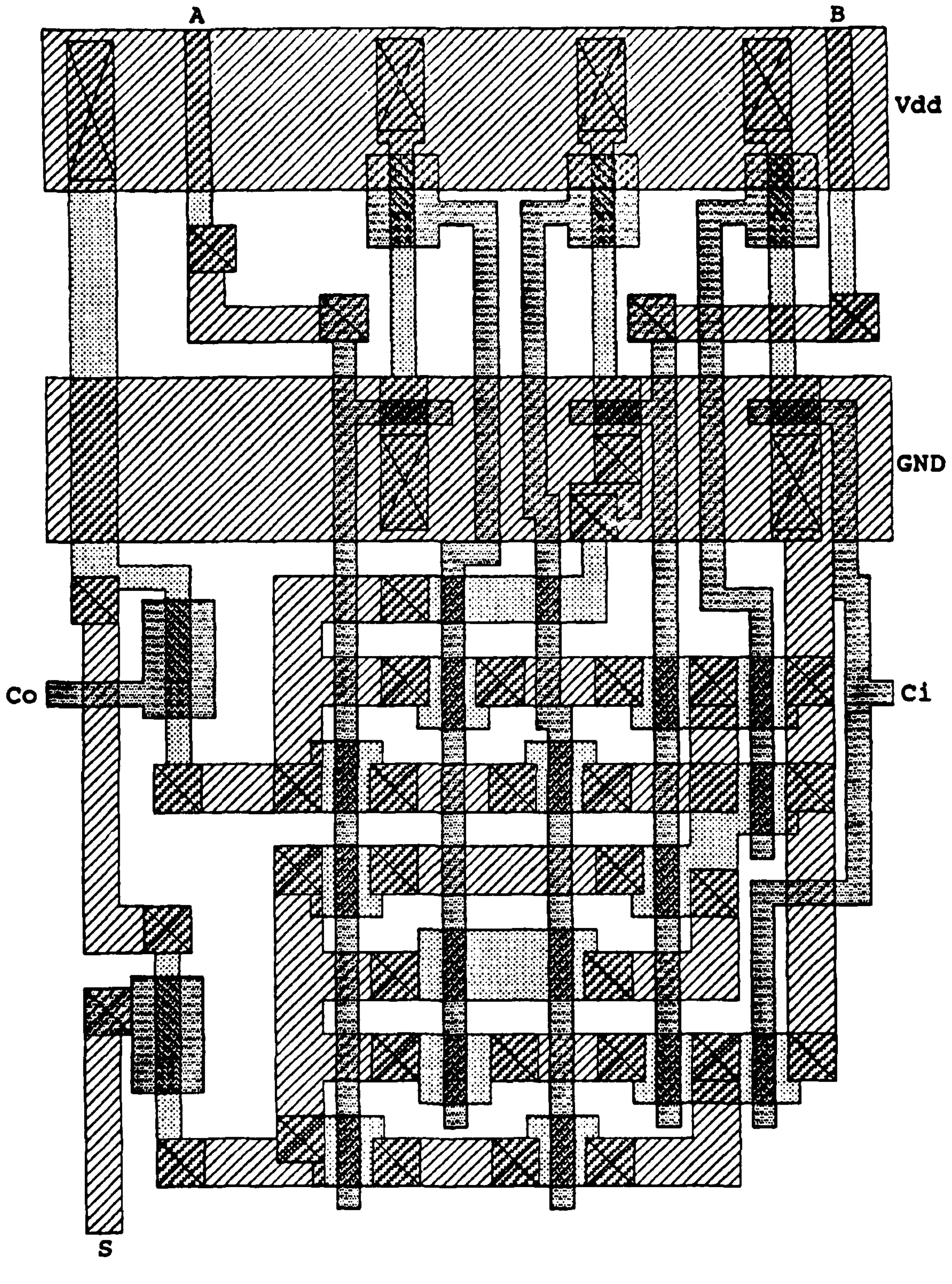


Figure 2.18 (c)

(c) 2220 Adder Layout

are several branches which may cause the output of this gate to go low; this implements the NOR part. However, there are several conditions that must be true in order for a particular path to be able to pull the output low; this implements the NAND part. Appropriate combinations of the two adder inputs, the carry input, and their complements act as inputs to these gates to form the sum and carry out functions.

This cell was constructed in a 71λ by 103λ area. Power busses stack from side to side. A and B inputs to the adder are available at the top of the cell, and the sum is output at the bottom of the cell. Carry in and carry out stack from side to side to allow fabrication of multibit adders. SPICE simulations give a worst case propagation delay of 48 ns and rise and fall times of 99 ns and 12 ns, respectively, for the sum output, and times of 41 ns, 94 ns, and 12 ns for propagation delay, rise time, and fall time, respectively, for the carry output, for output loads of 0.5 pf and at 100 degrees C. Figures 2.19 and 2.20 show graphical SPICE output for the sum and carry outputs.

The timing characteristics of this cell seem rather slow. It should be noted that the output load of 0.5 pf was arbitrarily selected for simulation purposes, and that actual loads (and therefore delay, rise, and fall times) will be smaller.

2.4.6 2300 Multiplier Cell

The last custom cell, the 2300 Guild multiplier cell, is depicted in Figure 2.21. This is simply a modification of the 2220 adder cell with the AND of the coefficient and data inputs going to one input of the adder and the partial sum input going t the other. This cell was constructed in a 71λ by 103λ area. Power busses stack from side to side. The data input feeds through from the top to the bottom of the cell for ease of interconnection when the multiplier array is formed. Likewise, the coefficient input feeds through the cell from side to side, and the carry input and carry output stack from side to side. The partial sum output is available at the bottom of the cell. SPICE simulations give a worst case propagation delay of 47 ns and rise and fall times of 99 ns and 12 ns, respectively, for the partial sum output, and times of 45 ns, 96 ns, and 12 ns for propagation delay, rise time, and fall time, respectively, for the carry output, for output loads of 0.5 pf and at 100 degrees C. Again, the output load was arbitrarily chosen for simulation purposes, and actual circuit performance will be more acceptable. Figures 2.22 and 2.23 show graphical SPICE output for the partial sum output and the carry output of this cell.

2.5 SYSTOLIC ARRAY FOR HIGHER ORDER FILTER

The custom cells described earlier were used to construct a ninth order filter chip. The design is contained in a 64 pin standard pad frame issued by the MOSIS fabrication facility and sealed for use in $\lambda = 1.5$ micron designs. Interior project dimensions were 7900 by 9200 microns. The design was built using an heirarchical approach to construct a network of cells that conform to the floorplan discussed previously. Figure 2.24 shows the cell organization of the ninth order chip with the required 5 slices of previously presented floorplan. Note that filter required only about half of the 7900 by 9200 micron frame. This structure is a classical example of a linear systolic array, with a number of identical processors designed for a specific application. Each processing element performs the same function simultaneously on different data, with data being exchanged with adjacent processing elements.

The ninth order chip was fabricated by the MOSIS facility and its performance is described in the next section.

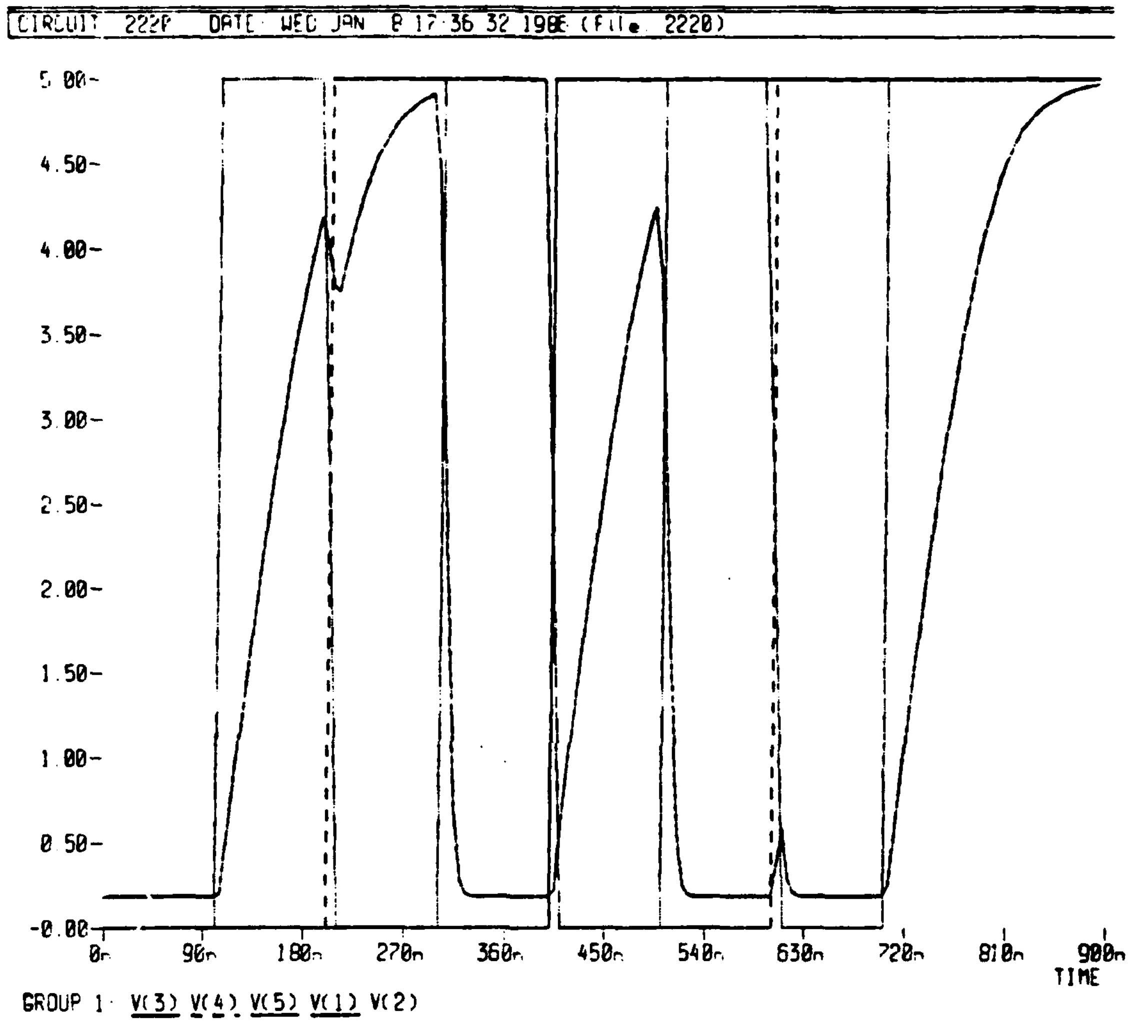


Figure 2.19

2220 Adder SPICE Output for the Sum Output

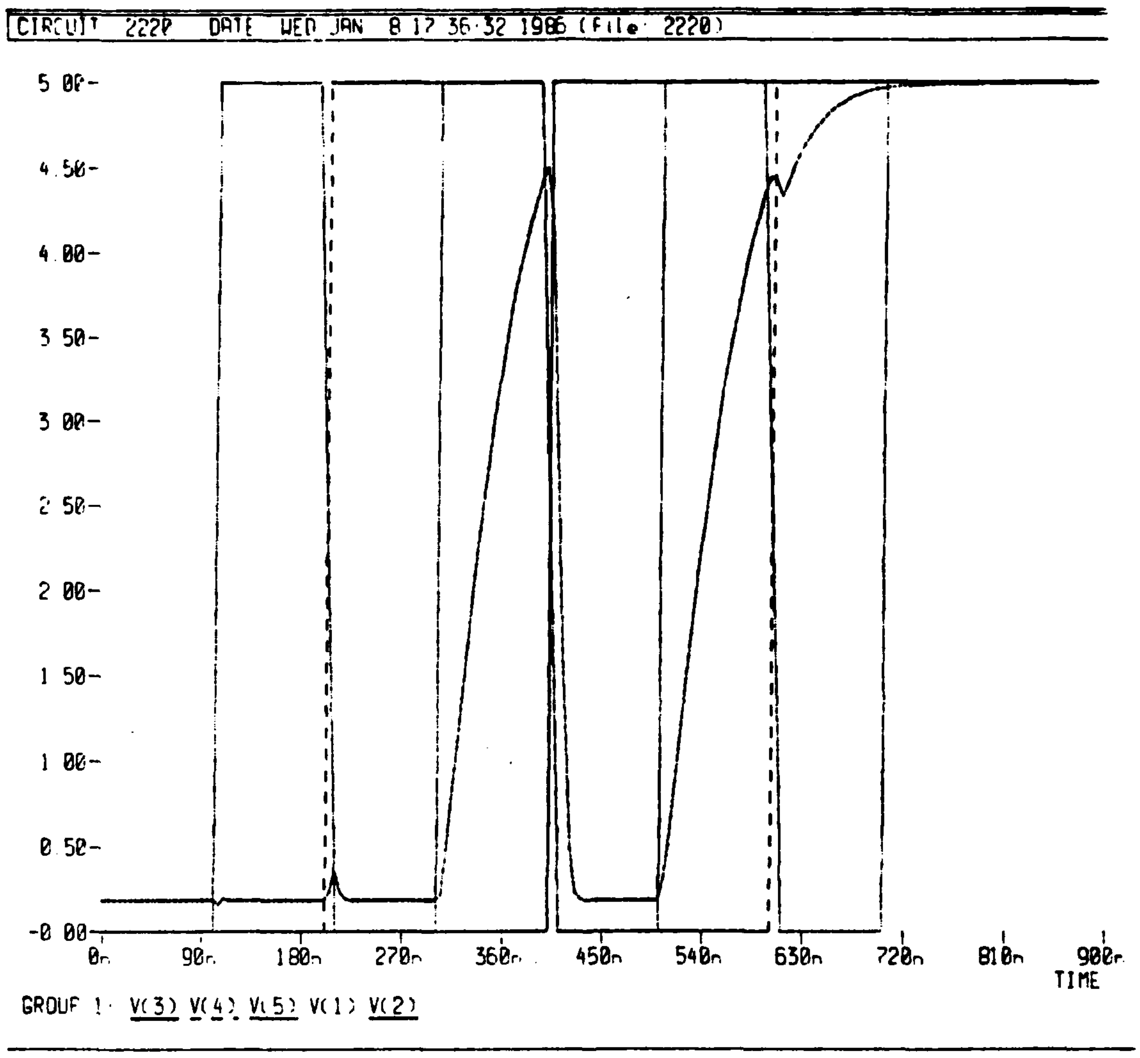
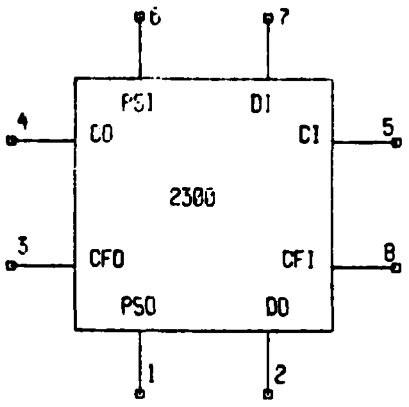


Figure 2.20

2220 Adder SPICE Output for the Carry Output



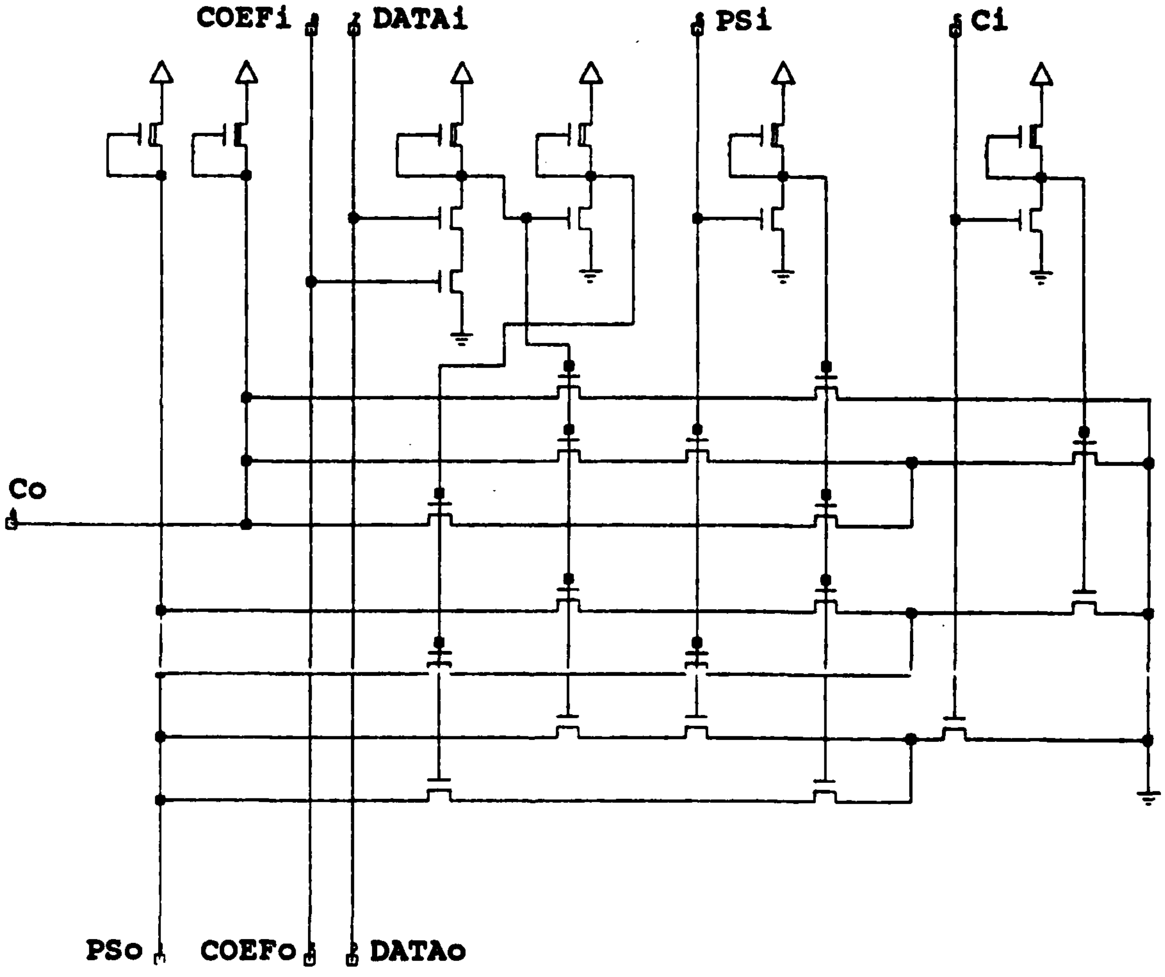


Figure 2.21 (a), (b)

(a) 2300 Multiplier Symbol
(b) 2300 Multiplier Schematic

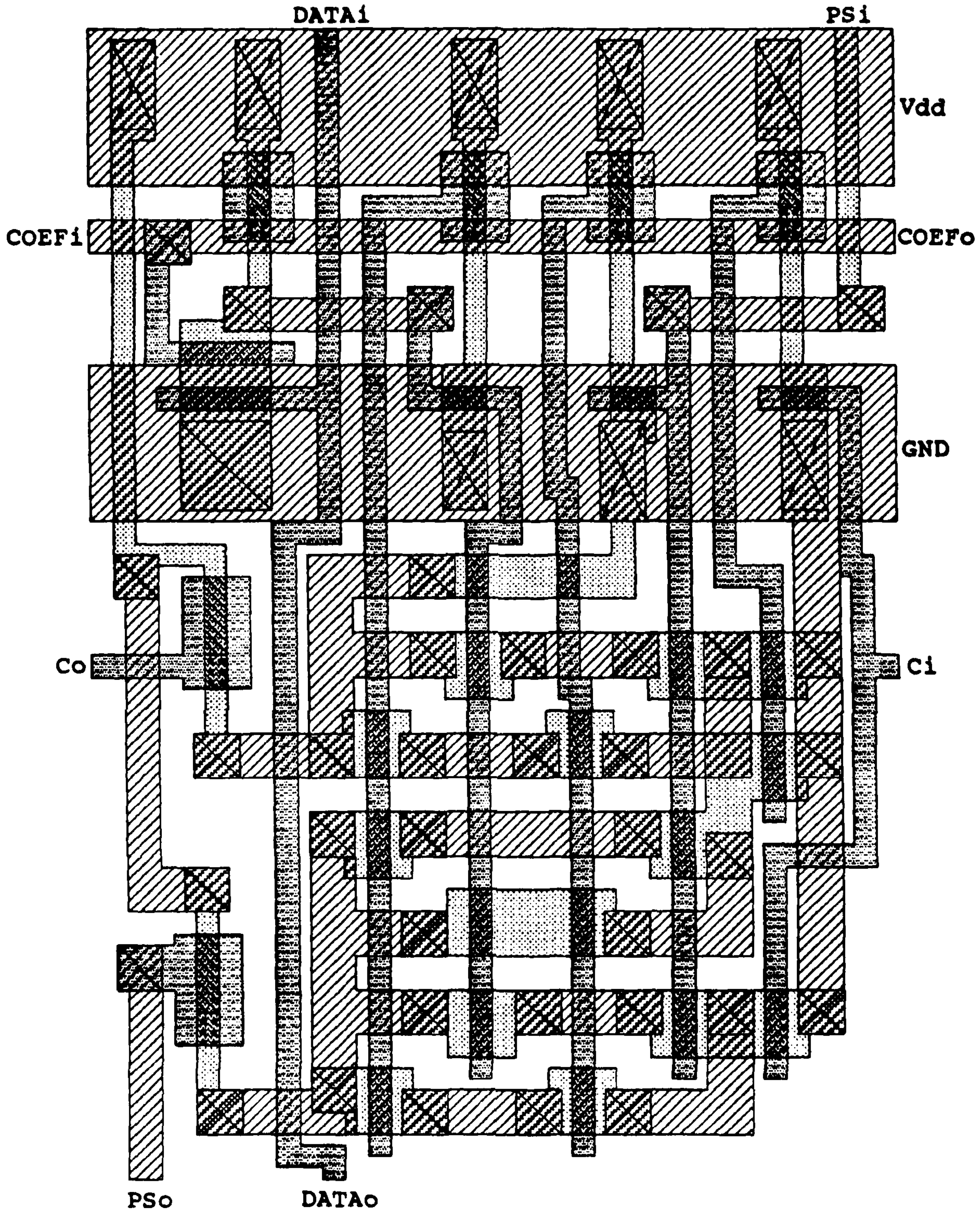


Figure 2.21 (c)

(c) 2300 Multiplier Layout

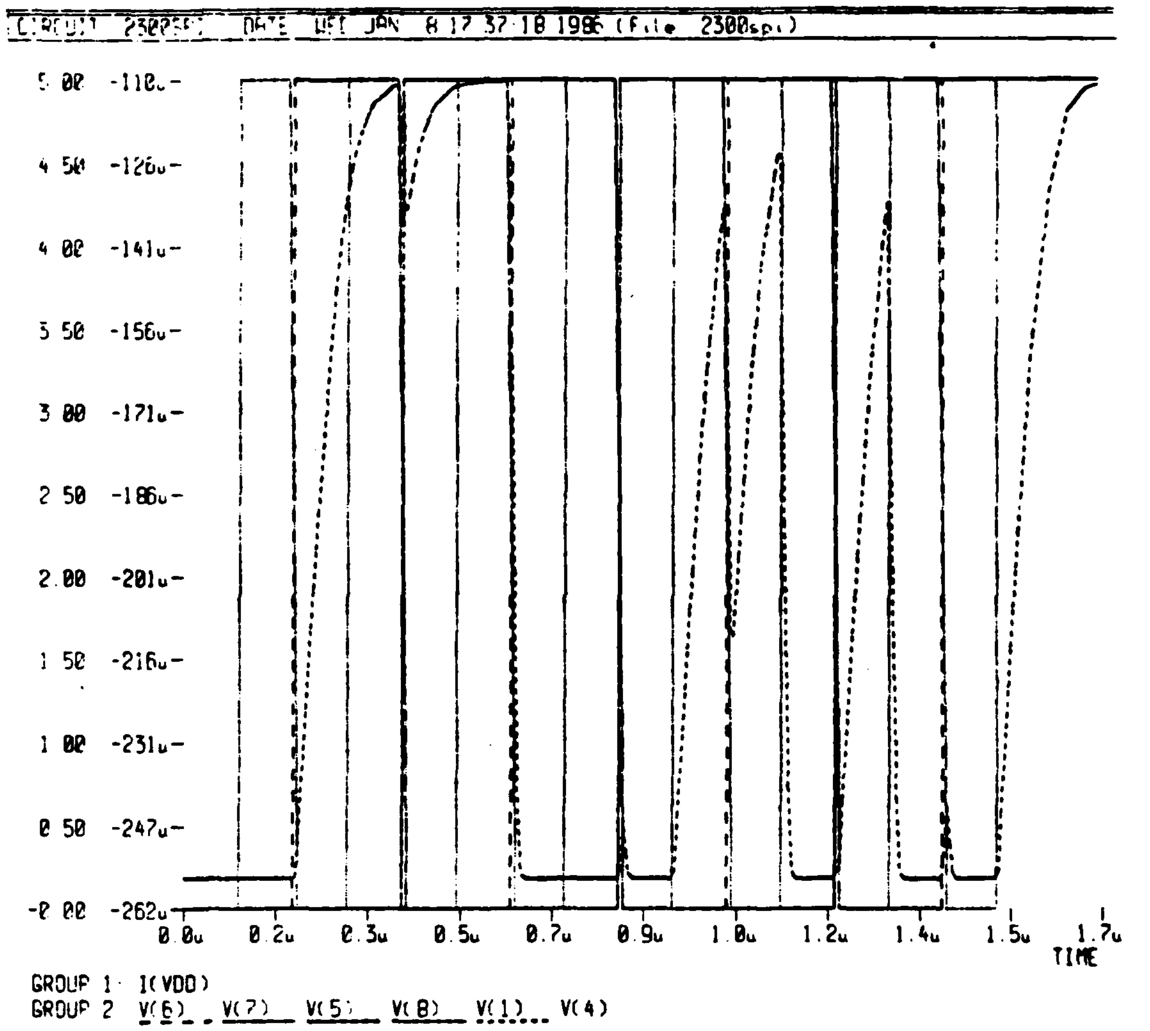


Figure 2.22

2300 Multiplier SPICE Ouput for the Partial Sum Output

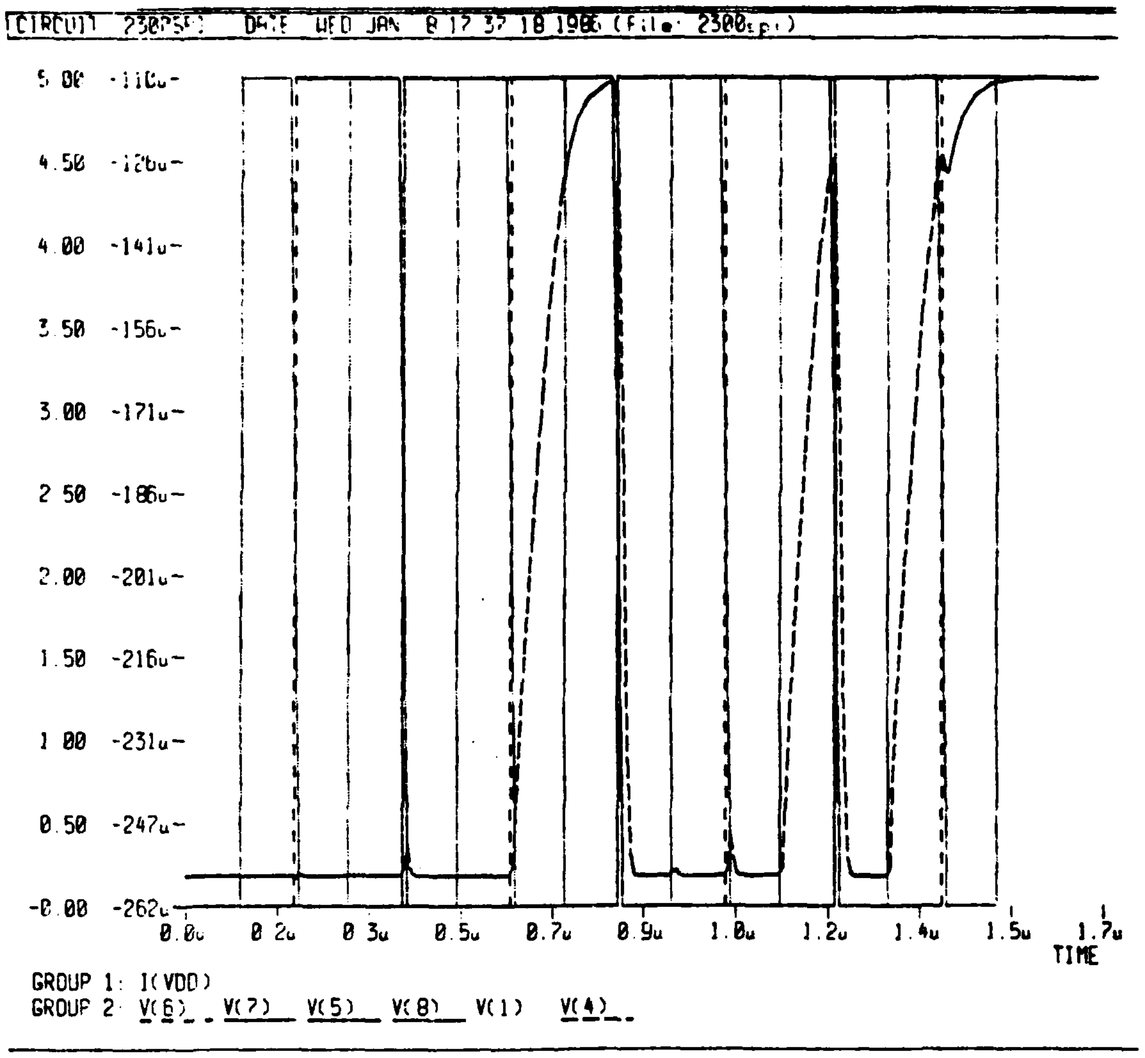


Figure 2.23

2300 Multiplier SPICE Output for the Carry Output

2.6 PERFORMANCE OF PIPELINED NINTH ORDER FILTER

The previously discussed FIR program was used to generate coefficients for testing the ninth order circuit. The basic test configuration is shown in Figure 2.26 where the highly concurrent processor consists of four cascaded ninth order filter circuits. Testing revealed a maximum clock frequency (sampling rate) of about 2 MHz. Using the previously mentioned algorithm, optional filter coefficients were generated with the objective of obtaining the most narrow bandpass filter with the four cascaded filters with at least 50 dB stopband rejection and at most 1 dB ripple in the passband. This was accomplished by specifying an ideal bandpass width of zero at the desired center frequency and iterating to find the minimum transition width which would give each ninth order circuit a 10 db stopband rejection with no more than .25 dB error at the center frequency. The response of the four cascaded circuits should be the product of each individual transfer function. The measured results for a typical center frequency which agree closely with the predicted performance are shown in Figure 2.26.

2.6.1 Multirate Sampling

A second method of pipelining filters uses two different sampling frequencies. The first two filters are clocked at one fourth the speed of the last two. The second set of two filters are programmed with coefficients that result in a bandpass filter with a center frequency f_{s1}. The first set of two filters is programmed to have a bandpass filter with a center frequency at any one of four multiples of the first; i.e., $f_{s1}, 2f_{s1}, 3f_{s1}$, or $4f_{s1}$. the first set of filters can yield an improved narrow passband because of the slower clock, and the second set of filters are then used to reject the unwanted aliased responses. Figure 2.27 shows the measured results obtained using this method for four pipelined ninth order filters. Note that other multiples for the slow clock could be used such as division by eight. This would give a better narrow band filter for the aliased response of the slower clocked circuit, but would require a sharper filter for the fast sampling frequency with poorer stopband rejection.

2.7 TASK I SUMMARY

A full custom FIR filter was designed and fabricated using NMOS technology. The result was a very compact design with moderate sampling rates. The design is based on a slice approach where the order of the filter can be extended by merely adding identical slices to create a systolic array. Constraints on the size of chip which could be fabricated resulted in a realization of ninth order filter for testing. The ninth order circuits were pipelined to obtain narrowband filters with good response characteristics. A multirate sampling technique was presented which significantly improves the narrowband filter obtained by the pipelined set of four ninth order circuits. The multirate sampling technique is a simple and practical method of improving the response of cascaded filters of arbitrary order.

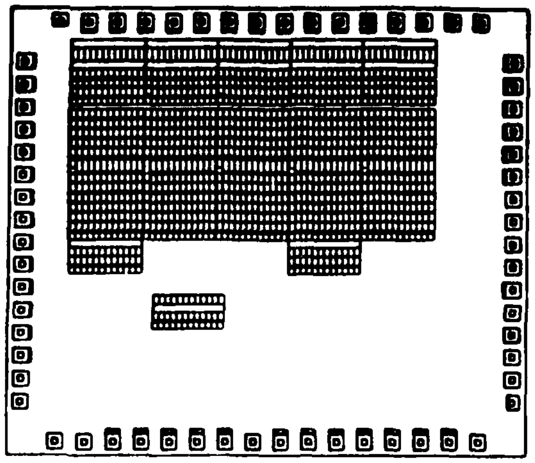


Figure 2.24

Five Slice Chip

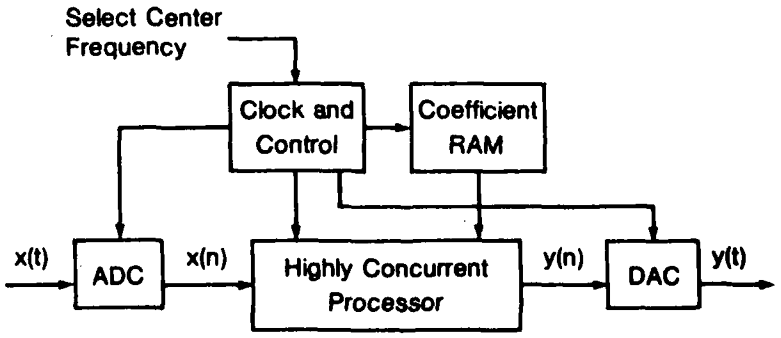


Figure 2.25

Test Circuit

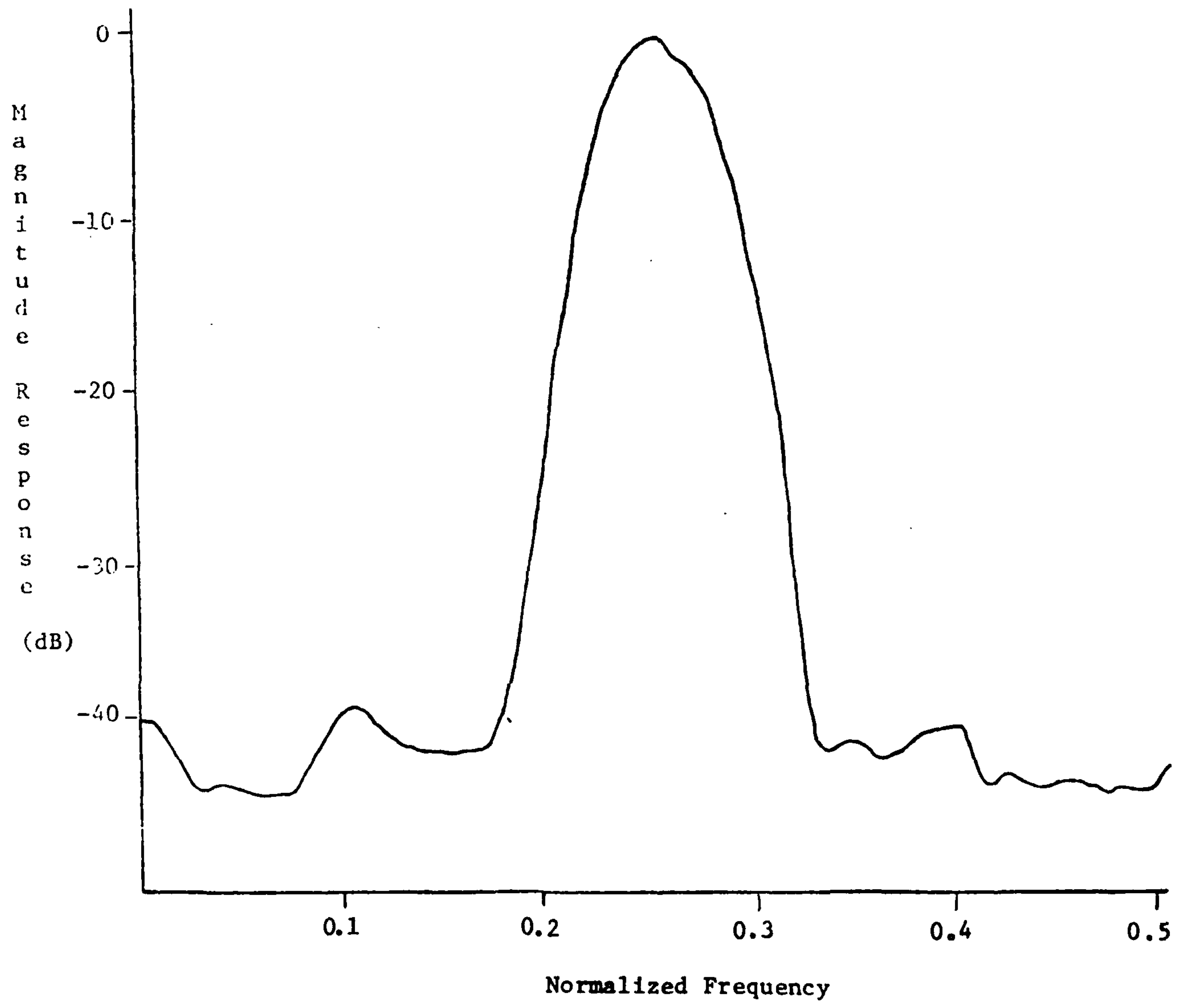


Figure 2.26

Pipelined Filter Response

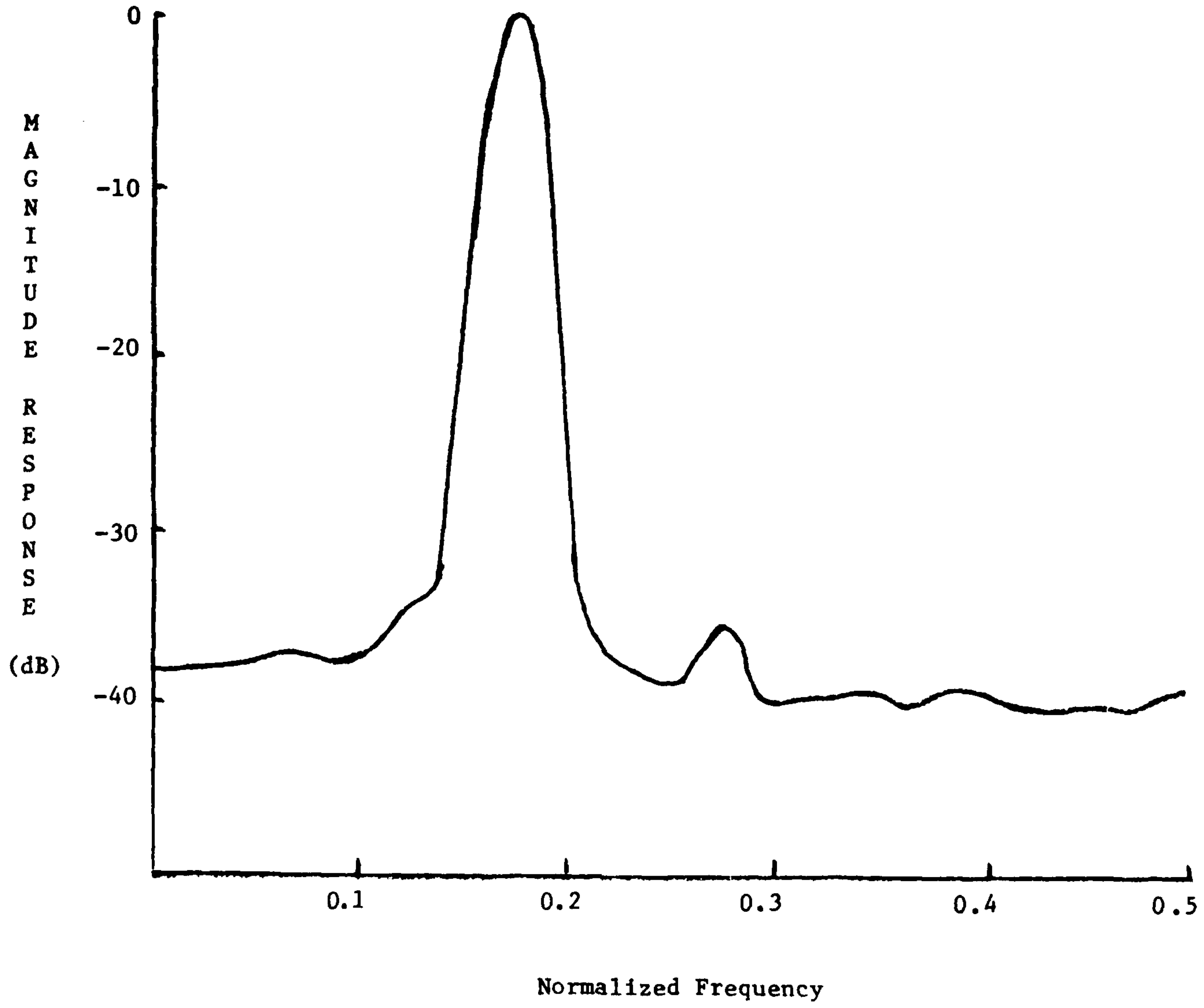


Figure 2.27

Multirate Sampling Response

Section 3

TASK II - PROGRAMMABLE DIGITAL FILTER USING CMOS VLSI TECHNOLOGY

The previous section discussed the design and performance of a FIT filter using nMOS technology. This task is an enhancement of that effort in that different digital design techniques of adding and multiplying are implemented using CMOS technology. CMOS has several advantages over nMOS. It has lower power dissipation and can have a higher clock rate than nMOS technology. The disadvantage of CMOS is its larger size due to the two types of transistors needed for each logic circuit [12]. However, the sizes of the transistors are decreasing. The feature size used in this project is 3.0*um*, and the results can be scaled to smaller sizes currently available.

3.1 ARCHITECTURE

The architectural techniques used for this programmable filter will be pipelining and parallel processing. In using the pipelining techniques, the fastest clock rate of the system is dependent on the slowest stage of the pipe. In this project, each pipe is broken up into a time rate of four carries. In other words, each stage is built so that the performance is based on four carries.

The design technique used is a hierarchical approach. The circuit is broken down into basic slices. The slices are further broken down into functional components which are further broken down into basic cells. The basic cells are standard cells found in the cell library which were obtained by the government sponsored MOS Implementation System (MOSIS) [13]. By reproducing these cells, the components can be built. These components are, then, interconnected together which creates the basic slices. By connecting the basic slices, the final circuit results.

Three tests can be performed on the project before it is sent out for fabrication. Vale, the computer–aided design tool used to develop the physical layout, contains a command that checks for design rule errors [14]. Spice is a dynamic simulator which helps determine voltage levels and rise, fall, and delay times [15]. Esim is a logic simulator which tests the correctness of the circuit [16]. These simulations are used on the project and are discussed in further detail in Section 3.3.

3.2 DESIGN APPROACH

3.2.1 Slice

The linear phase filter as discussed in Section 2 can be broken down into a reproducible slice, shown in Figure 3.1. The sample x(n–1) from the previous slice is added to the sample from the next slice. The sum is then multiplied by the weights h(m) to produce a partial result p(n). By adding up all the partial results, the output y(n) is obtained.

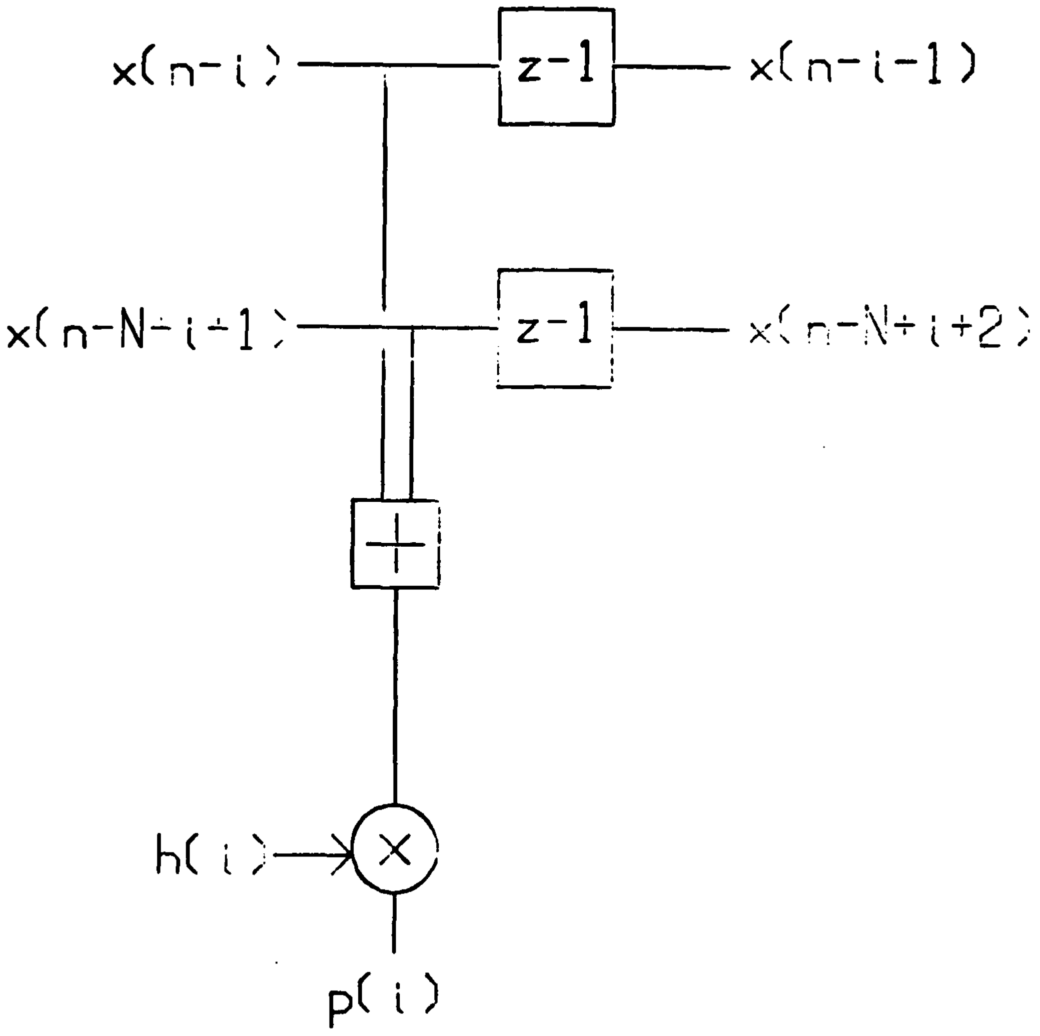


Figure 3.1

Reproducible Slice

Because of the size of the slice (5,005*um x* 2,132*um*) and the size of the frame (7,900*um x* 9,200*um*), only four slices fit in the frame. This means a single chip is a seventh order filter.

3.2.2 Floorplan

Figure 3.2 shows the basic floorplan of the slice. In the first row, x(n–i) and x(n–(N–i–1)) samples are stored in the dual sample latch in two's complement form. The samples are passed on to the next and past slices and are also added together in the second row. No overflow detection is provided. The data are then changed into one's complement preserving the sign. The sign is added to the one's complement data to obtain a number in two's complement positive form. This is done because the matrix multiplier can only handle positive numbers. The next 10 rows represent the multiplier and weighted coefficient latches. The coefficients are also stored in the sign–magnitude form with the sign preserved. The details of the multiplier are discussed later in section 3.2.3.4. The next 5 rows represent the product changing back into the original two's complement form. Also, in this set of rows, the next slice is added to the current slice. These rows are shared between every tow slices. The final latch, therefore, contains the output p(i) + p(i+1). Since the chip contains four slices, an extra adder and output latch are needed to produce the final product.

The size of the samples is 8 bits wide. This was chosen for various reasons such as the size of the slice, the ease of pipelining multiplication, and the availability of 8–bit A/D and D/A chips. The seventh order filter is large enough to realize practical filtering by cascading. There are various ways to cascade filters. One way is to cascade four chips to produce a seventh order filter in which the final data of one chip is passed to the next chip. This process is continued until the final data from the last chip is present. Another way to cascade chips, and the one of interest here, is cascading chips together to produce a higher order filter. For this section, four chips were cascaded together to produce a 31st order filter.

Eight bits also help break the multiplication by inserting three pipes into the multiplier. The first pipe is added after the first three carries. The next pipe is inserted after the next four carries, and the final pipe is inserted to hold the final product.

The final reason 8 bits are chosen is the availability of commercial 8–bit A/D and D/A chips that can run at the speed of the filter. The monolithic video A/D converter form TRW is a 20–MegaSample Per Second (MSPS) full parallel A/D capable of converting up to 7 MHz into 8–bit digital words. The video speed D/A converter also from TRW is a 20 MSPS D/A capable of converting 8–, 9–, or 10–bit data words. Both chips are compatible with TTL levels and can operate in two's complement data forms.

3.2.3 Components and Cells

Looking at the floorplan, there are four functional components which make up the filter: the latches, the carry–select adder, the exclusive OR complements, and the matrix multiplier. Each of these components is broken down into four reproducible cells: the latch cell, the full adder cell, the exclusive OR cell and the multiplier cell.

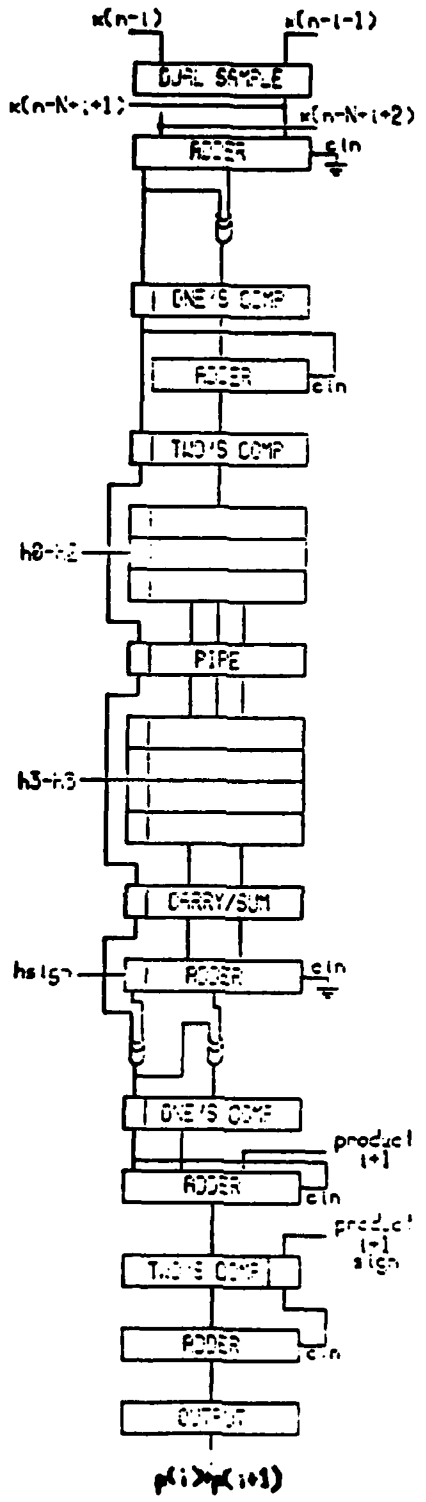


Figure 3.2

Floorplan

3.2.3.1 Latch

The latch cell, Figure 3.3, is of the D–type flipflop which uses the master/slave approach. The D input is clocked into the master when the clock is high. Then, when the clock goes low, the data is recycled through the master and also moves into the slave where the output Q and Q are presented. On the next high clock, new data is inserted in the master while the old data are recycling in the slave.

The one's and two's complement and the output latches are a single row data storing circuit which is made up of arrayed latch cells. The pipe latch contains three single latches which store the carry, sum, and multiplicand. The carry/sum latch contains two single latches which store the final carry and sum from the multiplier (see Fig. 3.4). The dual sample latch also contains two single latches which store the next and previous samples. This circuit contains an added feature of a 2–to–1 multiplexer (see Fig. 3.5). Recall that in the linear phase form, the last slice has only one sample and instead of passing the sample to the next slice, it passes the sample back to the previous slice. The mux selects which sample is to be passed to the previous slice (see Fig. 3.6). this circuitry permits one chip to be cascaded with another chip to create a higher order filter. For example, if each chip is a 7th order filter, by cascading two chips, a 15th order filter is obtained. In this design, four chips were cascaded together to obtain a 31st order filter. The last type of latch in the filter is the coefficient latch which is one bit long and stores the two's complement positive weight for the slice.

3.2.3.2 Carry Select Adder

The adder cell, Figure 3.7, is a full adder that forms two outputs from three inputs. A and B are the two input bits to be added and C is the carry from the previous adder. These inputs produce the outputs S and CO, that is the sum and the next carry. This cell can be arrayed to form the carry–select adder [18] shown in Figure 3.8. Bits a0–a3 are added to bits b0–b3 at the same time bits a4–a7 and b4–b7 are added twice, once with the carry low and once with the carry high. The carry from the fourth bit then selects the proper partial sum s4–s7 to go along with the rest of the sum s0–s3. This break up of sums complies with the four carry theory of trying to keep all the pipes as fast as possible.

3.2.3.3 Exclusive OR

The exclusive OR complement has natural characteristics that change the data, if necessary, from one's complement form to sign–magnitude form and from sign–magnitude form to one's complement form. As mentioned before, this is done because the multiplier can only handle positive numbers. The cell is shown in Figure 3.9 and is made up of two 'NAND' gates and one 'AND' gate. The inputs are A and B and they produce the result OUT.

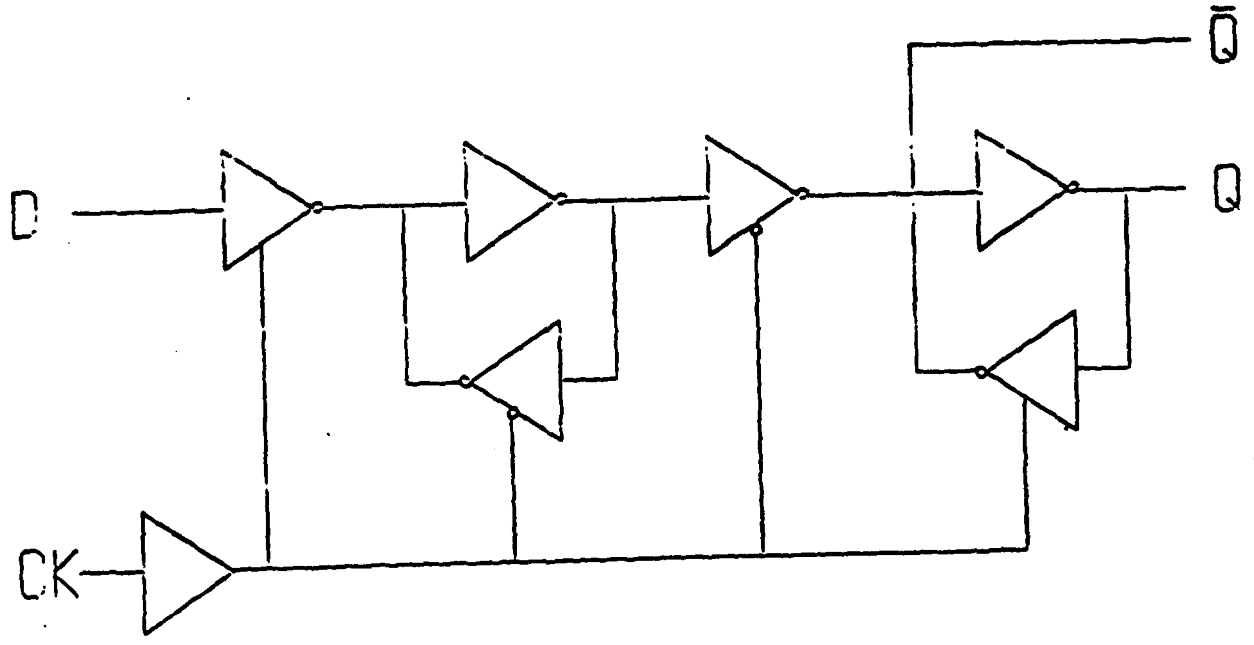
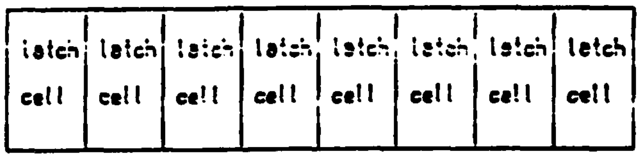
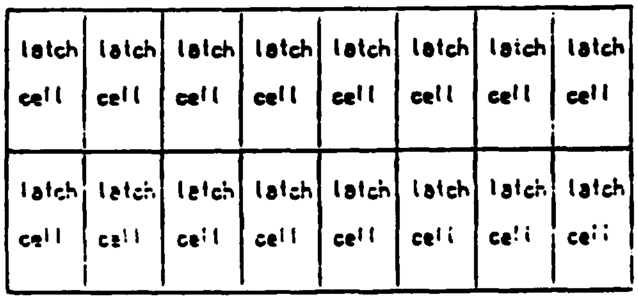


Figure 3.3

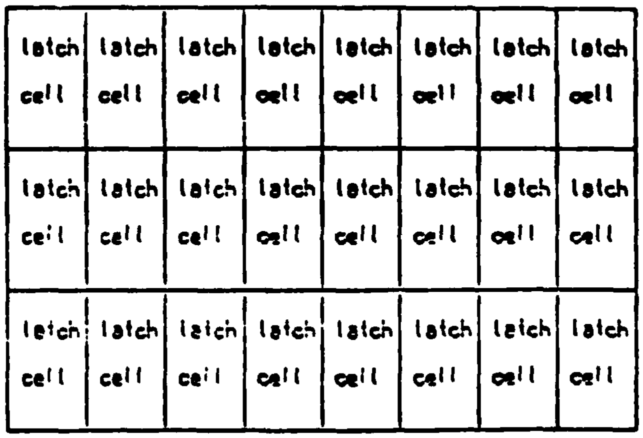
Latch Cell



one's complement
two's complement



carry/sum



pipe

Figure 3.4

Latch Components

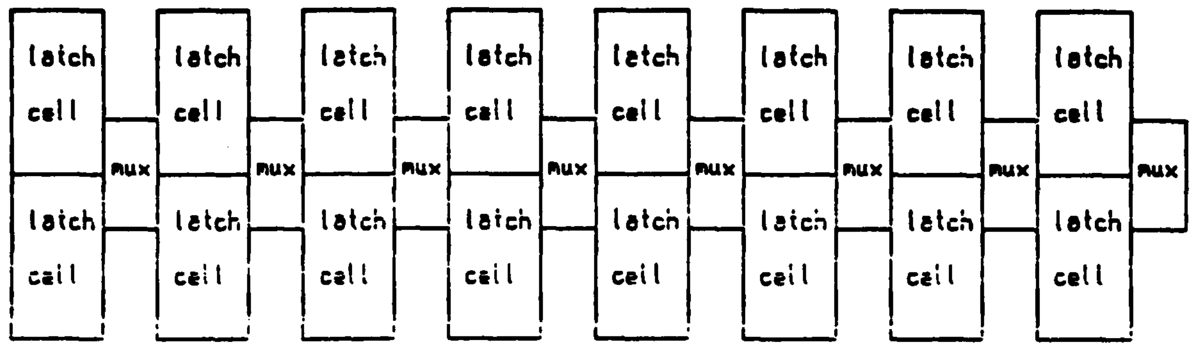


Figure 3.5

Dual Sample Latch

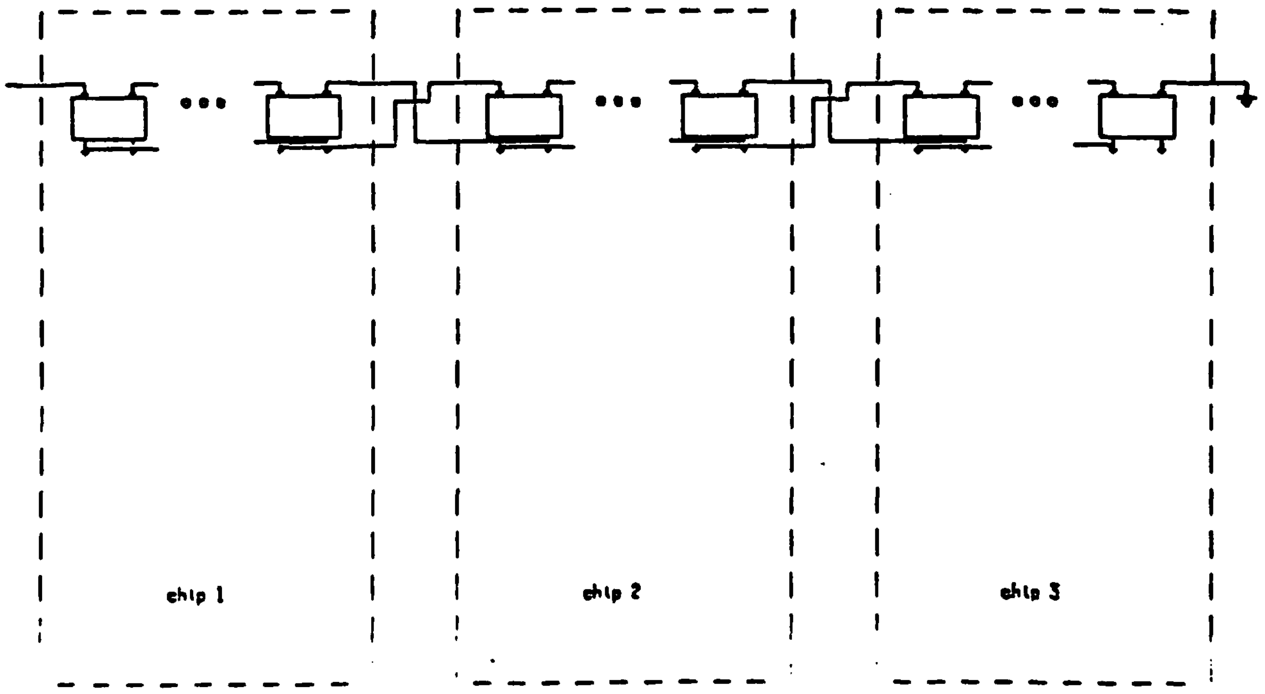


Figure 3.6

Chip Cascading

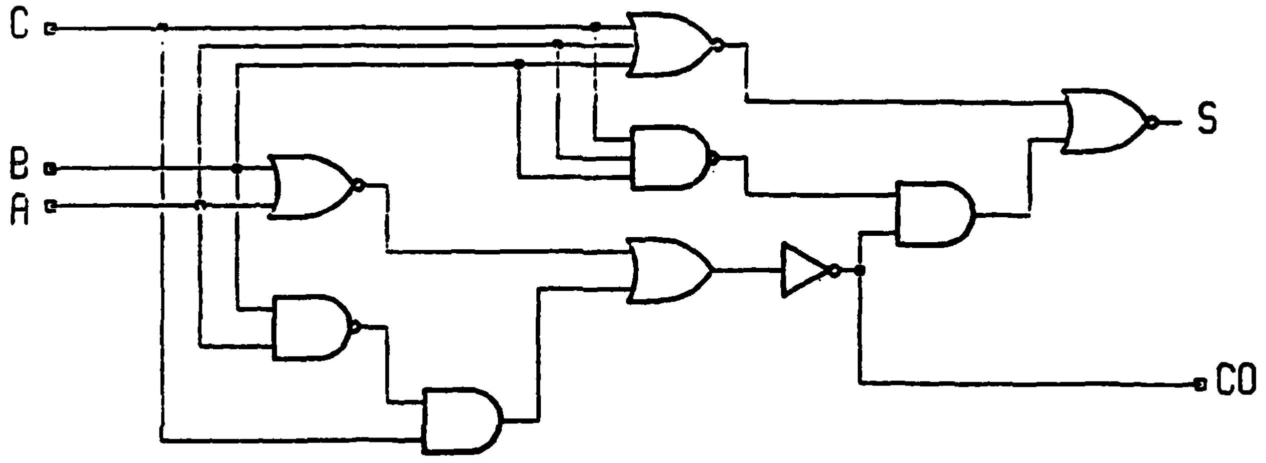


Figure 3.7

Adder Cell

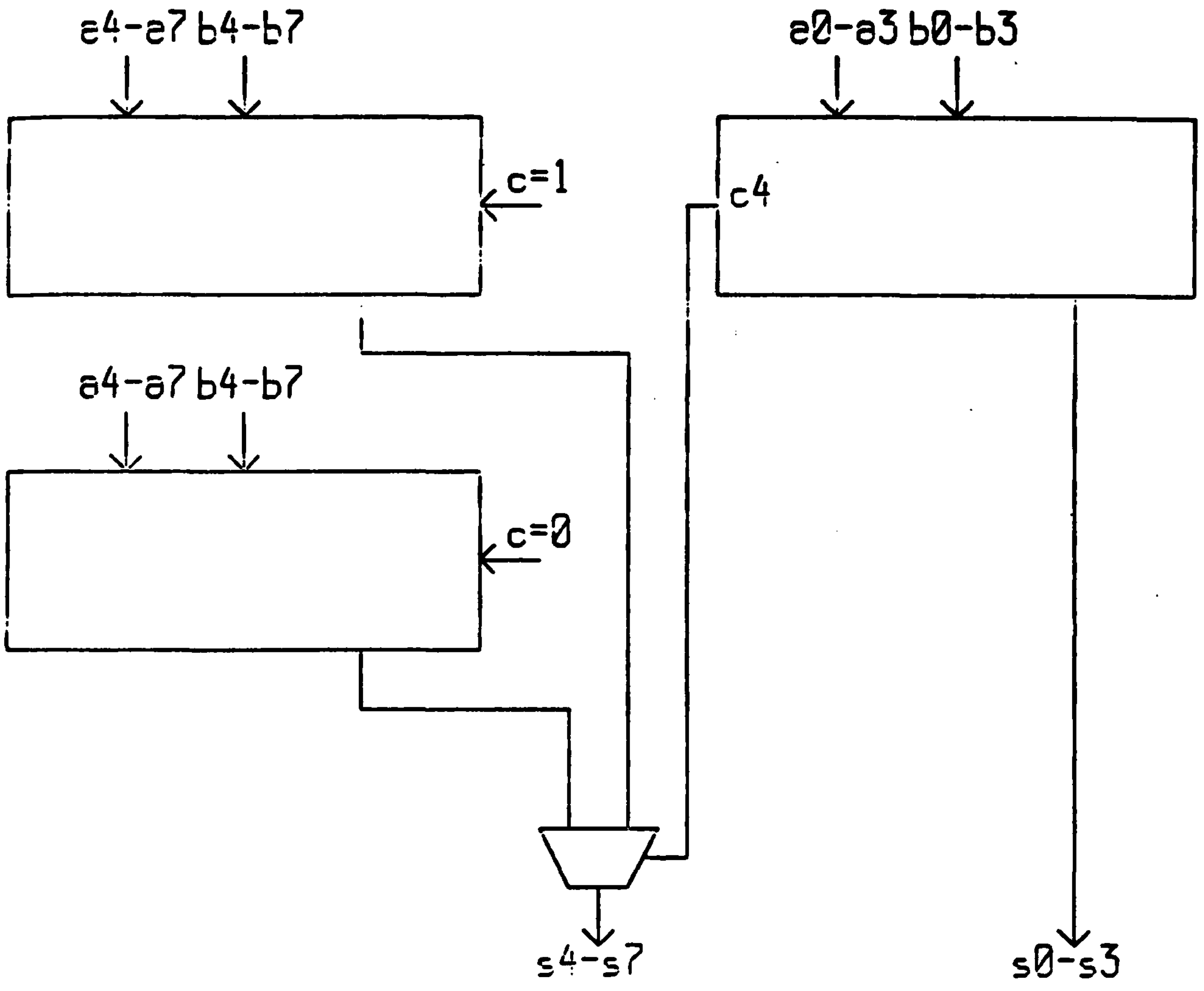


Figure 3.8

Carry–Select Adder

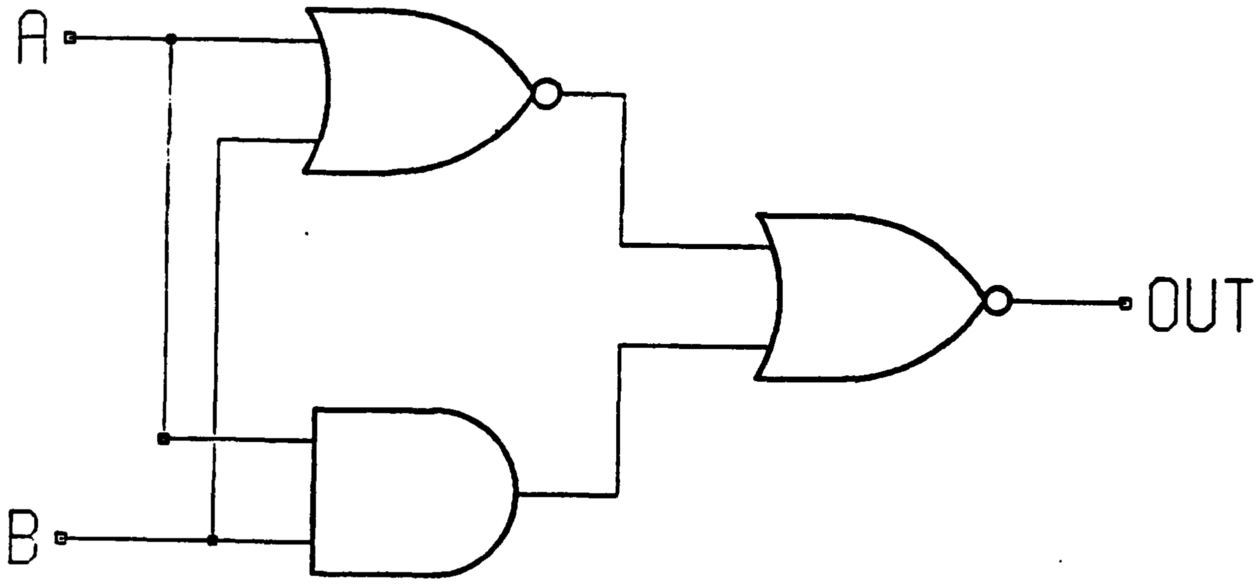


Figure 3.9

Exclusive OR Cell

3.2.3.4 Multiplier

First, it would be beneficial to go over how bits are multiplied.

```
    a1   a0
X   b1   b0
-----------
     a1b0 a0b0
a1b1 a0b1
-----------
p2   p1   p0
```

b0 is multiplied with a0 to produce a0b0, then b0 X a1 = a1b0, and so on. Then, by adding each partial product of each column together along with the carry from the previous column, the product is obtained. In this multiplier, each column is added up at the same time and whenever there is a carry produced by a two's column, it is added to the next highest two's column on the next add. A block diagram of the multiplier cell [18] can be seen in Figure 3.10. It has four inputs and produces two outputs. Multiplying inputs a(i) by b(j) to produce a(i)b(j) can be done easily by using an 'AND' gate. This result is then added to the partial sum, psi, and to the carry, ci, from the previous column which in turn produces the next carry, ci+1, for the next two's column and a partial sum, psi+1, for the next multiplier cell. The logic layout of the multiplier cell (Figure 3.11) is composed of the adder cell and the 'AND' gate. A better way to visualize the multiplier is to combine this cell into an example of a 4–bit X 4–bit multiplier (see Fig. 3.12). Notice, p0–p3 fall out and p4–p7 are made by adding the sums and carries. Once again, the multiplication is split by pipes to keep the timing to four carries. Also, a product of 14 bits is produced but in order to sustain the same output length as input length, the product is rounded to 7 bits word length.

3.3 SIMULATIONS

3.3.1 Spice

The following graph (Fig. 3.13) is an example of a spice simulation. This particular graph shows the delay for a one bit full adder. An estimated load of 1.5 pF was added to the sum and carry out to simulate the input capacitance of the next event. The delay time is calculated by subtracting the time when the input V(a) reaches 2.5 volts from the time when the carry out, V(cout), reaches 2.5 volts. In this case, the delay time is

$$47.484\text{ns} - 42.478\text{ns} = 5.006\text{ns}$$

The cells have different loading capacitances because each looks into a different input capacitance of the next event. These loading capacitances are for worst case situations.

Spice is used to find the total time to move data from one event to the next. The longest delay of a pipe stage is determined by adding up all the delay times of each event in that stage. In the slice, the longest total delay is the addition of carry and sum produced by the multiplier and the conversion to one's complement. The clock rate is found by the

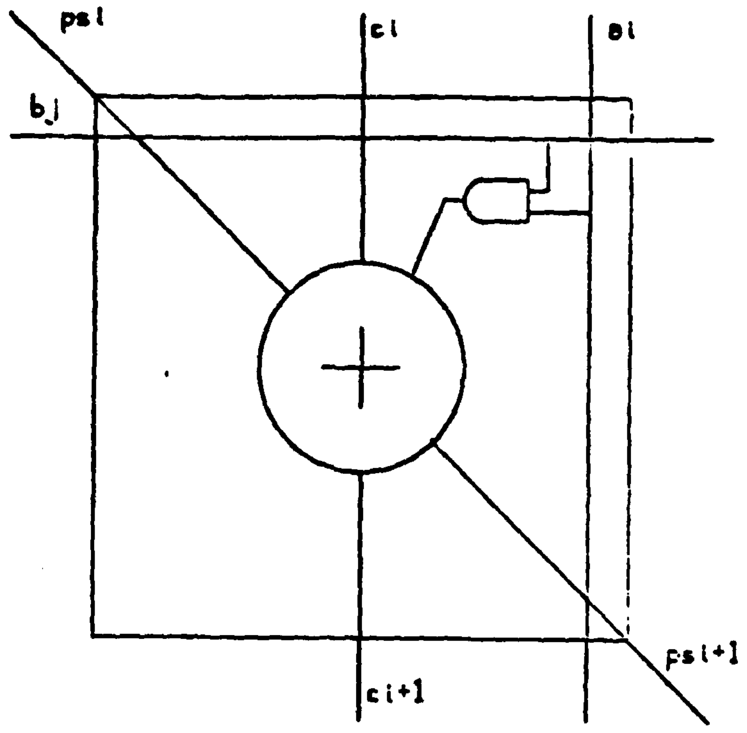


Figure 3.10

Multiplier Block Diagram

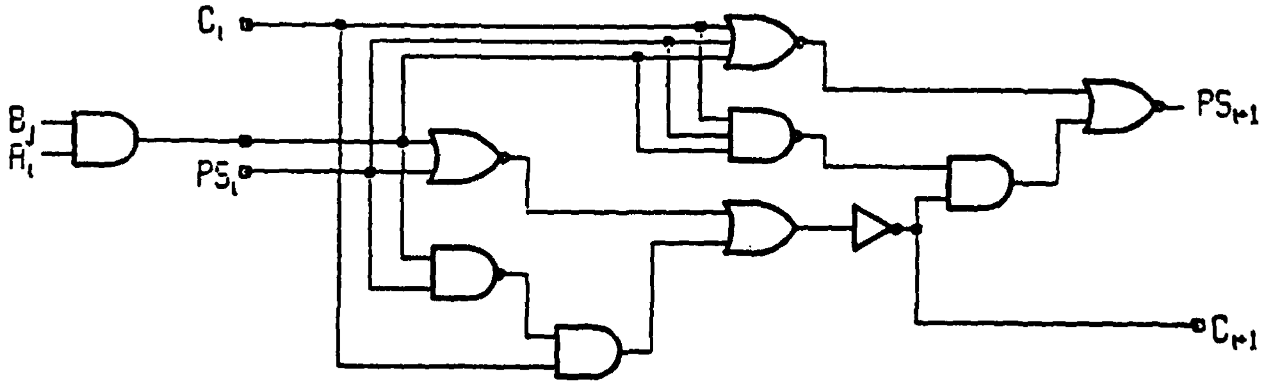


Figure 3.11

Multiplier Cell

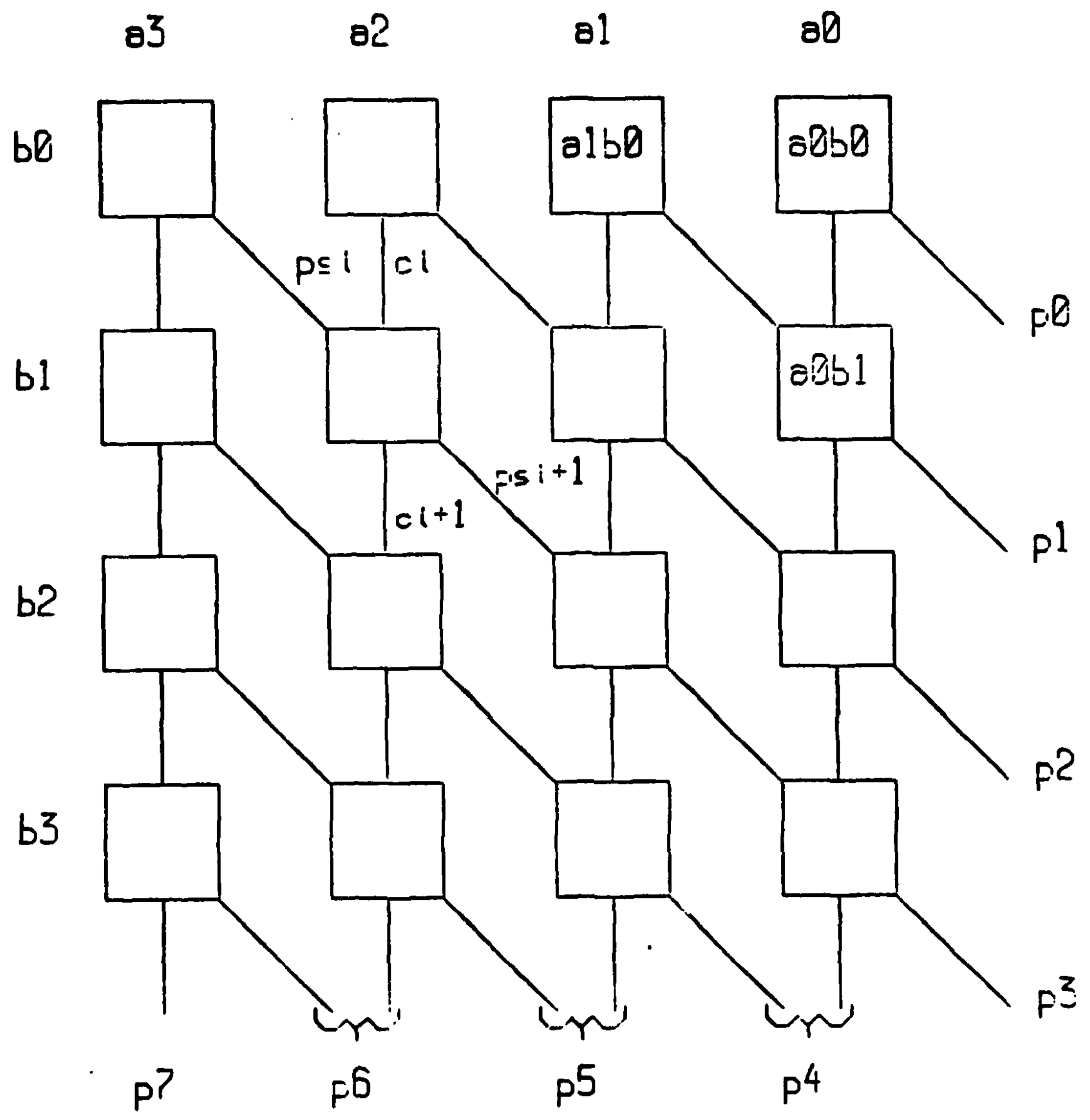


Figure 3.12

4–bit X 4–bit Multiplier

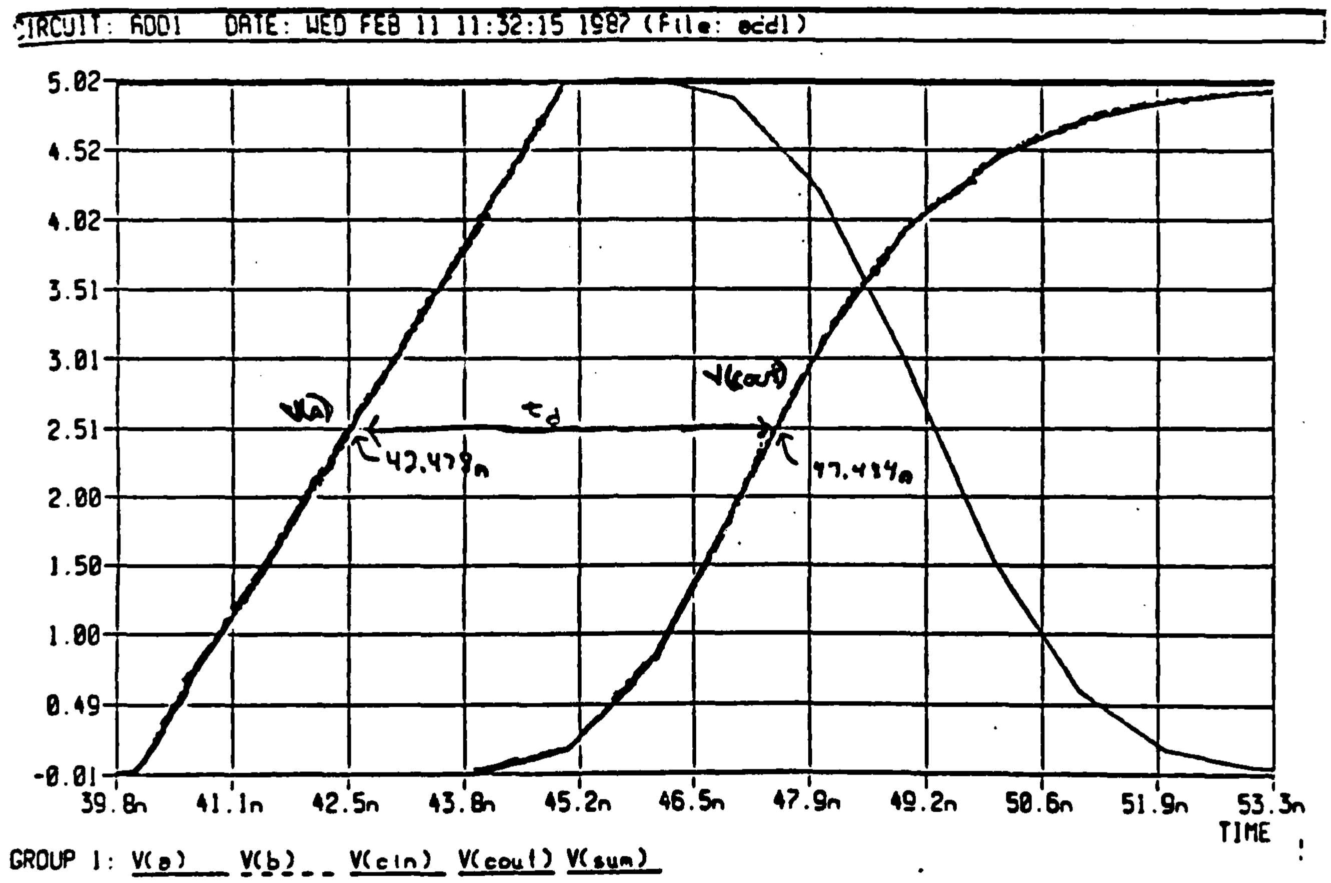


Figure 3.13

Spice Simulation

following delay times:

t(d) of latch = 6.44ns
t(d) of adder = 5.006ns
t(d) of exor = 4.33ns

By adding up the delays of each event, the total delay time is:

total delay = 6.44ns + (5.006ns * 4) + (4.33ns * 2)
= 35.124ns

The delay of the adder is multiplied by four because each carry–select adder has four carries and the delay of the exclusive OR is multiplied by two because the data travels through two exclusive OR gates. The clock rate is established by dividing the inverse of the delay time by two for a square wave.

clock rate = (1/total delay)/2
= (1/35.124ns)/2
= 14.23 MHz

Therefore, the chip should be able to clock at 14 MHz which is fast enough to handle the 0 to 3 MHz frequency range specification.

3.3.2 Esim

Esim simulations were performed on all cells, all components, and the complete slice. This helped build confidence that the circuit logically works. However, due to the large size of the circuit, a complete Esim study of the entire circuit was not performed because of memory problems on the Metheus, the system used to create the physical layout.

An example of an Esim simulation on a slice is shown in Figure 3.14. A0–A7 are the inputs from the next slice and they have a value of 25. d0–d7 with a value of 44 are the inputs from the previous slice, and b0–b7 are the coefficients with a value of –13. The next four inputs are the control lines. ck1 clocks the slice and ck2 clocks in the coefficients. The sel and seln control the 2–to–1 multiplexer on the dual sample latch. It is set as being the last slice in the filter, that is, d0–d7 is being sent back to the previous slice. The final control line is the load signal. When this line is high, the coefficients are loading into their latch on the next ck2 high pulse. Notice that the coefficients b0–b7 are changing but the final product p0–p7 is not. This is due to the load signal being low in the rest of the testing. the rest of the Esim simulations are the outputs corresponding to the inputs. l0–l7 represent the signal moving to the previous chip. These data are the same as the d0–d7 inputs which shows that the mux works as it is intended. r0–r7 are the outputs which would go to the next slice if there was one. a+do–a+d7 are check points that make sure the two sample inputs are added together correctly. pi0–pi7 represent the products of the multiplier before they are changed back to one's complement. Finally, p0–p7 is the one's complement product. In this case, the number is changed because the coefficient is negative. The two's complement product is not tested here because the next two stages of the floorplan are shared by every two slices.

```
4899 transistors, 2673 nodes (79 pulled up)
>11111111111111111111111111111111111111111111:a0
>00000000000000000000000000000000000000000000:a1
>00000000000000000000000000000000000000000000:a2
>11111111111111111111111111111111111111111111:a3
>11111111111111111111111111111111111111111111:a4
>00000000000000000000000000000000000000000000:a5
>00000000000000000000000000000000000000000000:a6
>00000000000000000000000000000000000000000000:a7
>00000000000000000000000000000000000000000000:d0
>00000000000000000000000000000000000000000000:d1
>11111111111111111111111111111111111111111111:d2
>11111111111111111111111111111111111111111111:d3
>00000000000000000000000000000000000000000000:d4
>11111111111111111111111111111111111111111111:d5
>00000000000000000000000000000000000000000000:d6
>00000000000000000000000000000000000000000000:d7
>11111111111111111100000000001111111111111111:b0
>00000000000000000000000000001111111111111111:b1
>11111111111111111111111111111111111111111111:b2
>11111111111111111111111111111111111111111111:b3
>00000000000000000000000000000000000000000000:b4
>00000000000000000011111111110000000000000000:b5
>00000000000000000000000000001111111111111111:b6
>11111111111111111100000000000000000000000000:b7
>0101010101010101010101010101010101010101010:ck1
>00000100000000000000000000000000000000000000:ck2
>00000000000000000000000000000000000000000000:sel
>11111111111111111111111111111111111111111111:seln
>00001110000000000000000000000000000000000000:load
>xx000000000000000000000000000000000000000000:l0
>xx000000000000000000000000000000000000000000:l1
>xx111111111111111111111111111111111111111111:l2
>xx111111111111111111111111111111111111111111:l3
>xx000000000000000000000000000000000000000000:l4
>xx111111111111111111111111111111111111111111:l5
>xx000000000000000000000000000000000000000000:l6
>xx000000000000000000000000000000000000000000:l7
>xx000000000000000000000000000000000000000000:r0
>xx000000000000000000000000000000000000000000:r1
>xx111111111111111111111111111111111111111111:r2
>xx111111111111111111111111111111111111111111:r3
>xx000000000000000000000000000000000000000000:r4
>xx111111111111111111111111111111111111111111:r5
>xx000000000000000000000000000000000000000000:r6
>xx000000000000000000000000000000000000000000:r7
```

Figure 3.14

Esim Simulation

```
>xxxxxx1111111111111111111111111111111111111:a+d0
>xxxxxx0000000000000000000000000000000000000:a+d1
>xxxxxx1111111111111111111111111111111111111:a+d2
>xxxxxx0000000000000000000000000000000000000:a+d3
>xxxxxx0000000000000000000000000000000000000:a+d4
>xxxxxx0000000000000000000000000000000000000:a+d5
>xxxxxx1111111111111111111111111111111111111:a+d6
>xxxxxx0000000000000000000000000000000000000:a+d7
>xxxxxxxxxx111111111111111111111111111111111:pi0
>xxxxxxxxxx111111111111111111111111111111111:pi1
>xxxxxxxxxx111111111111111111111111111111111:pi2
>xxxxxxxxxx000000000000000000000000000000000:pi3
>xxxxxxxxxx000000000000000000000000000000000:pi4
>xxxxxxxxxx000000000000000000000000000000000:pi5
>xxxxxxxxxx000000000000000000000000000000000:pi6
>xxxxxxxxxx000000000000000000000000000000000:pi7
>xxxxxxxxxxxx0000000000000000000000000000000:p0
>xxxxxxxxxxxx0000000000000000000000000000000:p1
>xxxxxxxxxxxx0000000000000000000000000000000:p2
>xxxxxxxxxxxx1111111111111111111111111111111:p3
>xxxxxxxxxxxx1111111111111111111111111111111:p4
>xxxxxxxxxxxx1111111111111111111111111111111:p5
>xxxxxxxxxxxx1111111111111111111111111111111:p6
>xxxxxxxxxxxx1111111111111111111111111111111:p7
4899 transistors, 2673 nodes (79 pulled up)
```

```
    a =  25         00011001
 +  d =  44       + 00101100
---------------   ----------------
  a+d =  69         01000101
 x  b = -13       x 10001101
---------------   ----------------
   pi = 897         00000111
   p  = -897        11111000
```

Figure 3.14 (continued)

Esim Simulation

3.4 CIRCUITRY

3.4.1 7th Order Chip

Figure 3.15 is a picture of the seventh order VLSI chip. Recursiveness and regularity of the hierarchical approach can be seen throughout the whole picture. The total size is 7,900*um* X 9,200*um* and contains 23,214 transistors. The chip is designed using the 3–micron rule set.

The pin numbers of the seventh order chip are as follows:

pin 1	–	Vdd substrate connection
pin 2	–	product 2
pin 3	–	product 1
pin 4	–	product 0
pin 5	–	coefficient clock 4
pin 6	–	nc
pin 7	–	clock
pin 8	–	Ground
pin 9	–	Vdd
pin 10	–	test point sign–coefficient
pin 11	–	test point coefficient 6
pin 12	–	test point coefficient5
pin 13	–	Load
pin 14	–	test point coefficient 4
pin 15	–	test point coefficient 3
pin 16	–	next output
pin 17	–	next output 6
pin 18	–	next output 5
pin 19	–	next output 4
pin 20	–	next output 3
pin 21	–	next output 2
pin 22	–	next output 1
pin 23	–	next output 0
pin 24	–	select
pin 25	–	next input 0
pin 26	–	next input 1
pin 27	–	next input 2
pin 28	–	next input 3
pin 29	–	next input 4
pin 30	–	next input 5
pin 31	–	next input 6
pin 32	–	next input 7
pin 33	–	previous input 0
pin 34	–	previous input 1
pin 35	–	previous input 2
pin 36	–	previous input 3
pin 37	–	previous input 4
pin 38	–	previous input 5
pin 39	–	previous input 6
pin 40	–	previous input 7
pin 41	–	previous output 0
pin 42	–	previous output 1

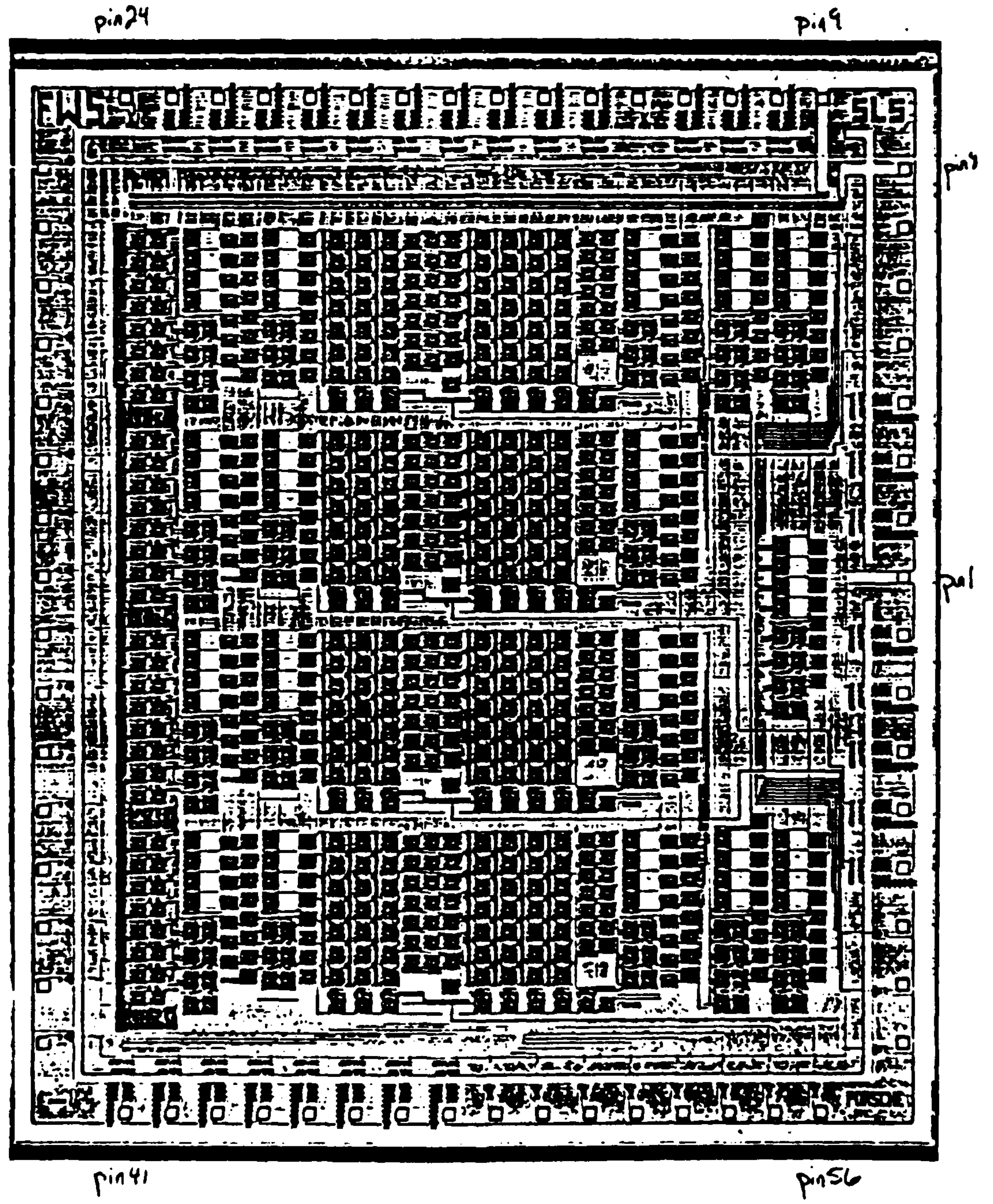


Figure 3.15

Physical Layout

pin 43	–	previous output 2
pin 44	–	previous output 3
pin 45	–	previous output 4
pin 46	–	previous output 5
pin 47	–	previous output 6
pin 48	–	previous output 7
pin 49	–	coefficient 0
pin 50	–	coefficient 1
pin 51	–	coefficient 2
pin 52	–	coefficient 3
pin 53	–	coefficient 4
pin 54	–	coefficient 5
pin 55	–	coefficient 6
pin 56	–	coefficient 7
pin 57	–	coefficient clock 1
pin 58	–	coefficient clock 2
pin 59	–	coefficient clock 3
pin 60	–	product 7
pin 61	–	product 6
pin 62	–	product 5
pin 63	–	product 4
pin 64	–	product 3

Something to notice in the picture is that the output pads have more detailed circuitry than the input pads. This is because the output pads have drivers in them to boost the output signals. Also, the last six rows of cells both on the right and left side add two slices together, and the sum of each of these is added together by the three rows in the middle to produce the final result y(n). Finally, there are three lines of metal on the right side which are used for Vdd, Ground, and clock line (from right to left).

3.4.2 Filter Board and A/D D/A Board

The seventh order chips are capable of filtering signals by themselves. However, to improve the responses, a filter board which cascades four chips together was designed to implement a 31st order filter. The digital input comes through a bus connector which is hooked up to the A/D D/A board. The data are then processed by the filtering chips, and the final result of each of the four chips is added together by TTL adders. This addition is also pipelined to keep up with the high performance clock rate. The final outcome is sent back through the bus connector to the A/D D/A board where it is changed back to an analog signal.

There is another bus connection on the filter board which is hooked up to the MC68230 parallel interface. The 68000 board aids in loading the coefficients [19].

3.5 PERFORMANCE EVALUATION OF 31ST ORDER FILTER

As previously discussed, the pipelined 31st Order Filter architecture should permit a clock frequency of 14 MHz and with the proper filter coefficients should result in narrowband filters with stopband rejection of –40 dB. Two problems surfaced during testing which inhibited full performance. The first problem involved the clock input circuitry. Testing revealed that the clock input line could be driven at a maximum rate of 1.5 MHz. Investigation of the clock input circuitry showed that all the clock lines internal to the chip were being driven through one unbuffered input pad which had a protection resistance of about 5K. The resistance, in conjunction with the capacitance of the on chip clock circuits (90pF), produces an RC time constant that limits the clock speed. A simple design using buffers would eliminate this problem. A second catastrophic problem was caused by skewing of the clock signal between the sign bits of the slices and the circuitry which adds the results of the slices. That is, the clock signal arrives at a slightly different time due to the different distance traveled from the input pad and results in a race condition when the input data crosses zero and switches signs. Again, this could be fixed with a buffer delay in the sign (it1) bit circuitry. These problems prevented complete testing of the filter. Testing at slower clock speeds with input data all the same sign indicated complete functionally otherwise.

3.6 TASK II SUMMARY

Task II resulted in the design of a pipelined processor which could be clocked in excess of 10MHz with minor corrections to the design. The 31st order filter circuit results in a systolic array of 16 processors performing 440 million operations per second (MOPS). The processor design incorporates an improved parallel architecture for the adder and multiplier circuits when compared to Task I. The fabrication was done using a 3μ CMOS process. Extrapolating the results to a 1μ process would indicate the same design would permit clock speeds in excess of 30 MHz and approximately one–ninth the area for a 7th order filter circuit.

Section 4

TASK III - A HIGH PERFORMANCE VECTOR PROCESSOR USING RESIDUE NUMBER THEORY, PIPELINING, AND VLSI TECHNOLOGY

Many algorithms and procedures have been used to increase the speed of arithmetic operations in digital computers. This section presents the use of an older algorithm (Residue Number Theory) combined with the use of newer technology, namely VLSI (Very Large Scale Integration). The design of a 16x16 bit vector processor is used as a design example in a general treatment of the implementation of processors using this technique.

4.1 THEORY AND NOTATION

The basic function behind Residue Number Theory is a simple one [22]. We take an arbitrary integer (positive or negative) which we will represent by b. This integer is then divided by another integer, which we will restrict to be positive for the purposes of this discussion. We will represent this integer by n. We define the residue representation Modulo n as the positive remainder of the division of b by n. The notation widely used in the literature is as follows:

$$R = (B)MOD(n)$$

where R is the positive integer remainder of the division of b by n.

One easy way to find R given b and n is to divide b by n, truncate the fractional part of the quotient, and find R by the following equation:

$$R = b - qn$$

where q is the truncated quotient of b and n, which is chosen such that R is a positive number.

The thrust behind a Residue Number System (RNS) is a tricky puzzle which may be illustrated by the following example:

> If we know that the residue representation of an unknown number u is 4 with respect to modulus 5, 6 with respect to modulus 7, and 2 with respect to modulus 3, what is the unknown number?

The answer of this puzzle is actually an infinite set of values separated in magnitude by 5x7x3, the product of the respective moduli. If we restrict u to be less than the product of the moduli, then every u in this set of values has a unique modular representation in the chosen modular system. This means that every u in this set will have a unique representation if the modular system representation of u is chosen to be an n–tuple with each value defined to be the modular value of u with respect to each modulus in the system. More compactly,

$$(R_1, R_2, \cdots, R_k) = (u)MOD(n_i),\ i = 1,2, \cdots, k.$$

As an aside, the structure of this puzzle is the format of the original CRT made nearly 2,000 years ago [20].

Recall that u is restricted to be less than the product of all the moduli in the system. Under ordinary circumstances, this restriction on u severely restricts the use of the residue number system in mathematical calculation. However, in digital computers, the restriction of the maximum magnitude of u is a natural consequence of finite word widths and poses no theoretical problems which would prevent the use of a Residue Number System (RNS) in digital processors.

The dynamic range of an RNS where each representation of u in RNS format maps back into a unique u is simply the product of the moduli. This fact does not hold in general, and is restricted to moduli chosen so that they are pairwise relatively prime. This restriction means that if we take all possible sets of the moduli two at a time, each pair only has unity as a common factor. Therefore, if we have a specific word size desired in a digital processor, we choose the moduli so that the dynamic range of these moduli is large enough to represent the largest number possible in the system. Other considerations enter into the selection of the moduli and will be discussed in later sections of this discussion.

What we are ultimately discussing here is a different representation for numerical data. One common representation for numerical data, the binary number system, is a fixed–radix system. This means that the representation of numbers depends upon a positional notation and a fixed base. A mixed–radix system is similar to a fixed–radix system, with each number depending on a positional notation, but with each position in the number dependent a unique base. The residue representation of a number is a radix–less representation which has an arbitrarily–chosen but fixed positional representation. One way to see the difference between a fixed or mixed–radix system and a radix–less system is the information conveyed in the format. A casual inspection of a number in any radix format usually yields an exact idea of how its magnitude compares with another number in the same format. However, numbers in a radix–less format cannot be compared in this fashion without a conversion of this number to some sort of radix system. This is one limitation to an RNS. This and other limitations to an RNS will be discussed in detail in later sections.

The choice of representation of numbers in a digital computer has a huge effect on the hardware realization of the processor. For example, if we are working on two different processors which will use binary format for numerical data with one processor using 2's compliment sign notation and the other using sign–magnitude representation for signed numbers, an adder system in each will be vastly different in each processor.

The consequence of using an RNS in a processor is a paralleling of the hardware used to perform arithmetic. The equation below defines the set of possible arithmetic operations while a number is in RNS format [23]:

$$\text{IF } (c_1,c_2,\cdots,c_n) = b_1 \text{ and } (d_1,d_2,\cdots,d_n) \text{ then,}$$
$$b_1 o b_2 = (c_1 o d_1.\ c_2 o d_2, \cdots, c_n o d_n) \text{ where o is } +,\ \text{x},\ \text{or} - .$$

A close inspection of this equation yields some useful properties and consequences of an RNS processor:

1) We have partitioned arithmetic into n parallel, independent, carry–free pieces.

2) Since the width of each part in the n–tuple of the RNS representation of u can be represented in binary by numbers much smaller in width than the original binary representation of u, arithmetic can be performed far faster while the number is in RNS format since the delay for arithmetic operations is proportional to the width of the operands.

3) The hardware used is the same to implement arithmetic operations independent of the sign of the numbers involved.

From the above, it is apparent that the main advantage of an RNS system in a digital processor is speed. It will be shown later that the delay for multiplication, which traditionally takes far longer than addition or subtraction in a digital processor using binary data, is roughly the same as for addition in an RNS processor. These advantages can be extremely important to the designer of a high–speed processor needing these qualities.

4.1.1 RNS Processor Research

Early attempts were made to construct RNS processors in the 1960's. These attempts, although successful, were constrained by the digital hardware technology of the time [21]. Researchers into the 1970's enjoyed a better technological advantage, but the costs of integrating custom hardware were very prohibitive for research to progress rapidly. Most advances in the 1970's concerned algorithm manipulation in order to use off–the–shelf integrated circuits and programmable logic devices with enhanced performance [23] [25]. These circuits tended to be very large and difficult to troubleshoot, and their performance was not as good as other binary processors of the time.

With the advent of VLSI design in the late 1970's, the hardware realization of RNS processors finally became practical [28]. However, costs were still very high at that time, so only well–endowed institutions could afford to constructprototypes. Research centered on algorithm manipulation, with the needs of VLSI technology in mind. Most hardware actually constructed consisted of only functional elements such as adders, with very few fully–functional processors constructed. Hughes constructed an RNS processor chip set in NMOS technology in 1979 with a dynamic range of 18 bits [33]. To date, most functional processor efforts can be categorized by word lengths of 16 bits or less, for use in specific filter architectures.

It is believed that the processor described in this section is the first fully functional 32–bit RNS processor that has been integrated onto a single chip. To achieve this, some unique hardware construction methods were employed. These methods will be discussed in later sections. This processor is not a stripped–down minimally functional one; it contains many other features, including the ability to handle signed of unsigned numbers, which is another unique achievement.

4.2 PROCESSOR DESIGN

4.2.1 Choice of Moduli

The choice of moduli in an RNS is an important one and is one of the biggest influences on system performance. Many different choices can be made, but some rules specific for the VLSI implementation of an RNS processor can be generally made and will be discussed in detail.

In the specification of a processor, the desired width of the output word is a basic quality of the processor. If this is known, then the dynamic range of the residue system is specified so that the size and number of the moduli in the RNS can be chosen so that this dynamic range is met. This does not eliminate many choices of systems of moduli, since an infinite combination of pairwise relatively prime numbers can meet this dynamic range specification.

The VLSI implementation of basic cells used in an RNS processor are generally composed of PLA's (Programmable Logic Arrays) [29] [30]. If great speed is desired inside the processor, the choice of smaller moduli will decrease the delay time through each of these cells. The modular representation of a number is actually an n–tuple of binary numbers in an RNS processor. Since each element of the n–tuple is always less than the respective modulus, the choice of smaller moduli will decrease the width of the elements in the n–tuple. Since the delay through a PLA is proportional to its size, and the size is proportional to the number of product terms, reducing the field width of each element in the RNS representation of numbers will enhance the speed of the processor.

Another consideration related to the width of the moduli is the desire in VLSI for all the basic cells in a design to have the same size and shape. Therefore, we want the width of the moduli in an RNS to be small and have the same width, for a regular, easily–planned layout which will have fast delay times. However, this may not be possible for systems with a large dynamic range, since it may be impossible for a pairwise relatively prime residue system to have all its elements the same width.

The last general requirement of the system of moduli chosen is associated with the conversion of the RNS number to binary. It turns out that the multiplicative inverse (discussed later) does not exist for every modulus in the RNS in general [21]. The existence of the multiplicative inverse is guaranteed if each modulus in the system is a prime number. The existence of the multiplicative inverse is required for the conversion from RNS to binary, so the choice of nothing but prime numbers will assure that the processor will avoid this existence problem.

In summary, when selecting the moduli in a RNS, choose the smallest–width prime numbers which will give you the desired dynamic range of the processor. This selection process should give numbers which are close to the same width for regularity in the lay out. If nothing but prime numbers are chosen, the system is automatically pairwise relatively prime.

For the processor described in this thesis, the desired dynamic range is 32 bits. This processor, termed the PRVP (Pipelined Residue Vector Processor) for purposes of discussion, performs a multiply–accumulate on 16 bit operands with 32 bits of accuracy. Using the guidelines just discussed, the following moduli were chosen for this processor:

$$(31,29,23,19,17,13,11,7)$$

The positions of each of the moduli in the n–tuple will be consistent with this positional notation throughout this thesis.

Notice that the width of each element in the n–tuple varies from five to three bits. This is due to the large dynamic range of the required system. Prime numbers with field widths of six bits could have been chosen so that the width of each field would be the same. However, this system would be slower than the one chosen.

The inclusion of an additional modulus could be implemented to facilitate sign detection while the number is in RNS format. All operations performed in the processor would be also performed on this additional modulus also, which is termed the redundant modulus. A check to see if this redundant modulus is even or odd can, under certain circumstances, detect the sign of an RNS number. Overflow can also be detected in this manner also, under certain circumstances. Since neither of these features was required for the PRVP, they were not included in the design. Since this technique is highly dependent on the moduli system, the desire to detect sign or overflow can govern the selection of the moduli.

4.2.2 Conversion Hardware

The magnitude of a binary number is represented by the following formula:

$$M = 2^n b_n + \cdots + 2^2 b_2 + 2b_1 + b_0$$

Where M is the magnitude and the b's are the binary number expressed from the most significant position to the least significant position. Since binary numbers are represented in this form, the modular representation of this number can be obtained by the following formula:

$$(x)\mathrm{mod}(m_i) = (2^n)\mathrm{mod}(m_i)b_n + \cdots + (2^2)\mathrm{mod}(m_i)b_2 + (2)\mathrm{mod}(m_i)b_1 + (b_0)\mathrm{mod}(m_i)$$

Where x is the modular form of the binary number (b) [21]. The implications of this formula are very important to the designer. As mentioned earlier, much of the hardware in an RNS processor can be composed of PLA's. This means that for every bit in b, we simply calculate the modular representation of the positional multiplier and perform a logical AND with the bit at the same time using a PLA and then add each output of the PLA's up with modulo adders. Of course, this approach would be very slow with many residue adder delays added sequentially. We could group some of the input field together and perform the table look–up on more than one bit of b. Since the delay through a PLA is proportional to the number of its product terms, we could balance the delays through the look–up stage and the adder stages in order to optimize the speed of the process. In particular, we would want to split b into c pieces, with c a power of two so that the number of residue adders used is minimized with the choice of c dependent on the resulting delays through the look–up and adder stages. Therefore, finding c is an iterative, trial and error process which depends on the width of each field of b and the width of the resultant residue representation of b.

The hardware realization of a residue adder will be discussed now. We could, of course, simply construct a PLA with inputs with w the width of the residue field. However, we could use a hybrid structure, where we use binary adders in the traditional way and then use an input PLA to scale the answer back to the proper residue representation [30]. We need the PLA, because in general the output of the adder may be larger than the modulus of that particular piece and must be converted back to the correct residue representation. The advantage of the hybrid approach is that it may be smaller than the direct PLA look–up approach without sacrificing much speed. Another advantage is that the PLA can do more than one thing, such as correct the output of the binary adders and adjust the sign of the number (discussed below). If a nonzero power of 2 is chosen for one of the moduli, we can eliminate the PLA look–up altogether. However, a power of 2 is not a prime number, and may cause problems in the convert–back process were the multiplicative inverse is required.

The PRVP has input operands of 16 bits. This input may be in either unsigned or 2's compliment format. After a few design iterations, the approach of Fig. 4.1 was used. The 16 bit number was broken up into two 8 bit fields and run through the PLA look–up tables. The output of these two tables was added together using a traditional ripple–carry adder structure. The output of the adder was adjusted by a PLA connected to the output. This PLA also converted the residue representation of b to a signed one if it was desired.

This sign conversion works as follows: 2's compliment format can be converted to a decimal number by subtracting from the magnitude of the binary number if the most significant bit is set and the word is k bits wide. Due to the distributed nature of the conversion from binary to residue, we can convert the number to residue regardless of the sign, and simply add the following to the number at the end of the process:

$$(-2^k)\mathrm{mod}(m_i)$$

Therefore, the PLA at the end of the hybrid adder can scale the number back to the correct residue representation and correct the sign of the number if that format is what is desired.

If we took the strictly look-up approach for the conversion to residue representation, we would need to add an additional PLA at the end of the process to accomplish sign compensation. However, if a hybrid residue adder is used instead, we can do the sign compensation at a cost of one additional input term in the PLA portion of the adder. This will be generally worth the additional space required for the PLA.

4.2.3 PLA's

We have been discussing PLA's for some time now without going into detail how they are constructed. MAGIC (layout editor) [32] has some associated tools with it which automatically generates the layout of PLA's given a truth-table description of the input and output functions. Table 4.1 gives an example of a truth table with five inputs and five outputs. The statements proceeded by a. in the description are directives for both the PLA generating program MPLA and the logic minimization program ESPRESSO [32]. The .I and .O directives tells each program how many inputs and outputs there are. The .phase statement indicates either true (1) or complimented (0) outputs for each of the output lines. Therefore, if you want the first output to have the inverse of what the truth table description indicates, we would change the first one to zero in the phase statement. This makes it handy to buffer the outputs of the PLA only using a single inverter.

ESPRESSO is run with this file (Table 4.1) as its input. ESPRESSO then minimizes the number of minterms required to implement each of the outputs. Each minterm requires an additional row in the PLA; therefore we are minimizing the size of the resultant PLA. It is not uncommon for ESPRESSO to reduce the number of minterms by half.

If the designer has outputs which are not required for a given input term, ESPRESSO can be used to further reduce the number of minterms. Instead of specifying a logic 1 or a logic 0 for this undesired output, a – (don't care) can be specified which will be used by ESPRESSO to its advantage. By an intelligent choice of inputs and outputs, a minimum version of a desired logical function can be realized without too much effort.

The output of the ESPRESSO program is a PLA description. This description is used by MPLA [32] to automatically generate the lay out of the PLA, which is based on a model stored in the file system.

The use of these programs can greatly speed the design of a VLSI circuit. The PLA descriptions can be automatically generated by custom software. This is easy when there is a mathematical relationship between the inputs and the outputs of the PLA, which is the case in an RNS processor. This will be discussed in greater detail shortly.

4.2.4 Conversion Hardware Details

After describing the design process for PLA's, we can now discuss the hardware in much more detail for the conversion to residue.

Table 4.1

ESPRESSO Input File

```
.i 5
.o 5
.p 32
.phase 11111
.type fr
00000  00000
00001  10010
00010  01101
00011  -----
00100  01111
00101  10010
00110  00011
00111  10101
01000  11111
01001  00000
01010  ----1
01011  00111
01100  10101
01101  00110
01110  11000
01111  -----
10000  11000
10001  11010
10010  10111
10011  00100
10100  -----
10101  01110
10110  00000
10111  -1---
11000  11010
11001  01100
11010  10001
11011  11111
11100  10001
11101  11100
11110  10011
11111  10011
.e
```

A version of Fig. 4.1 must be constructed for each moduli in the system. This means that for the PRVP, we must construct two PLA's per moduli, or a total of 16 PLA's. Since each of these PLA's have the same number of inputs and we wish to generate outputs for every possible minterm, at first glance we may conclude that each of these PLA's will be roughly the same size. However, the logical minimization ESPRESSO performs is not uniform for different moduli, and the sizes of the PLA's varied about thirty percent. As VLSI designers prefer to group functional blocks together in rows so that they may share common inputs, this approach was used. However, laying the PLA's out in rows would waste some space since some of the PLA's are smaller than other PLA's.

Originally, the use look-up adders in the converter was planned since these adders will be used elsewhere, which would cut design time. However, this approach was abandoned because the space wasted by this row of PLA's was just large enough to fit a set of hybrid adders into the space. Fig. 4.2 shows the modulo-29 stage of the conversion process. Fig. 4.3 shows the entire PRVP converter with the PLA's and the adders highlighted. Fig. 4.3 shows that, the modulo-29 stage contains the largest PLA's and also has a 5 bit ripple carry adder in the hybrid stage and hence is the slowest part of the conversion process. At the bottom of the highlighted area, we can see the sign and scaling PLA's for each moduli.

The Microsoft QuickBASIC programs used to generate the conversion look-up tables and the sign and scaling tables are contained [36].

4.2.5 Topology of the PRVP

The conversion hardware just discussed is very generic for many types of RNS systems. The next piece of hardware designed, the arithmetic unit (AU), is more specific for each application.

The specification called for a processor which could perform a multiply-accumulation operation at a very high speed with 16-bit operands and 32 bits of output accuracy. This processor was specified to interface with the SF1 (Stack Frame 1) Reduced Instruction Set Computer (RISC) on the high-speed stack bus. The SF1 is a processor designed using VLSI technology at Wright State University for high-speed 32-bit digital processing applications in a real-time avionics environment [22]. A processor such as the PRVP could provide a vector-processing capability which would be a natural extension of the SF1's ALU operations.

A block diagram of the PRVP is shown in Fig. 4.4. This block diagram includes the instruction set for the PRVP which are received via the Chip Select Bus (instruction bus) of the stack bus. From a study of Fig. 4.4, the architecture of the PRVP is pipelined and is very similar at this level of abstraction to a conventional binary processor.

Operands come into the PRVP and go through the binary to RNS conversion process via the stack data bus, which is actually 32 bits wide. The operand is latched into either A register or B register in signed or unsigned format (depending on a bit in the status register). This level constitutes one pipe. If the A register is loaded, it is implied that the product of the contents of the A register and the B register are added with the C register, which is the second pipe level. Thus, the C register keeps a running tally of the multiplications performed on the contents of A and B. When it is desired to know the value of C, the PRVP ca⁻ ˙e issued an instruction which will convert C into the correct binary format (signed o unsigned). At the end of this conversion process, the PRVP either generates an interrupt or sets a bit in the status register which tells the SF1 that the desired result is ready. While the PRVP is performing a conversion from RNS to binary, it can still perform other operations in the instruction set, such as loading new operands into the chip, clearing the C register, etc. Thus, a very high throughput can be achieved with the PRVP with an intelligent ordering of the instructions.

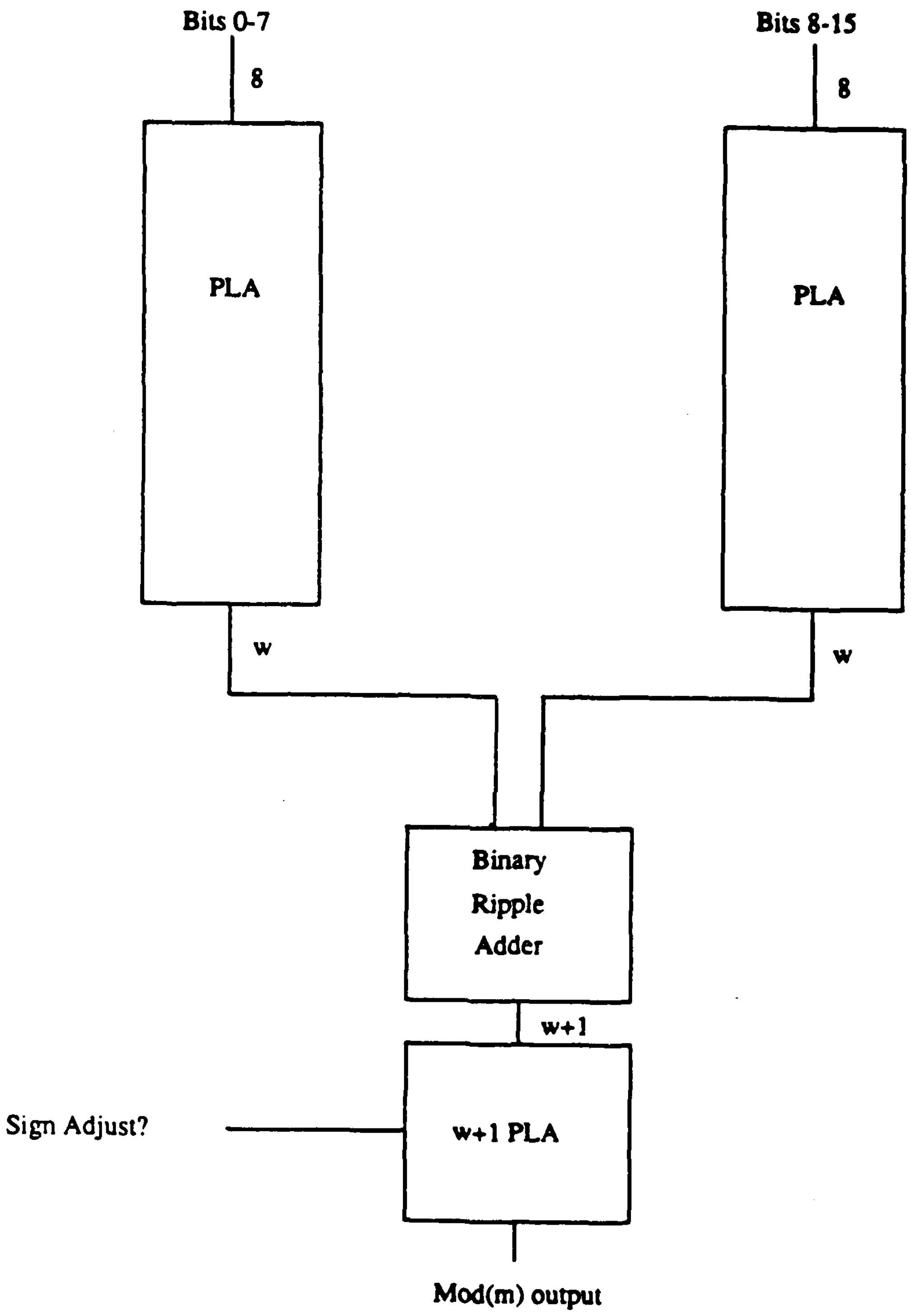


Figure 4.1

Block Diagram of the Binary to RNS Hardware

Figure 4.2

Mod–29 PLA's

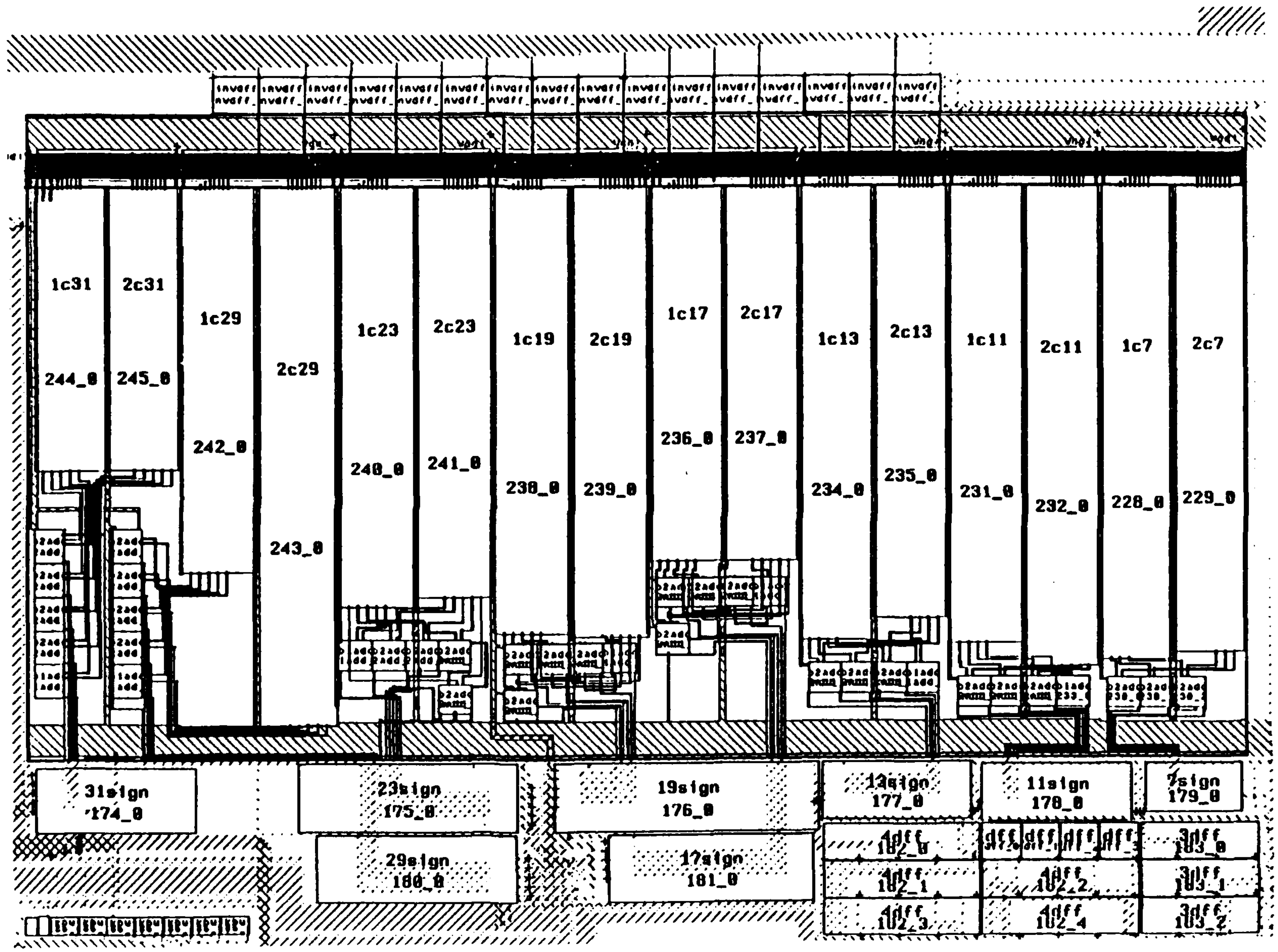


Figure 4.3

Layout of the Binary to RNS Converter

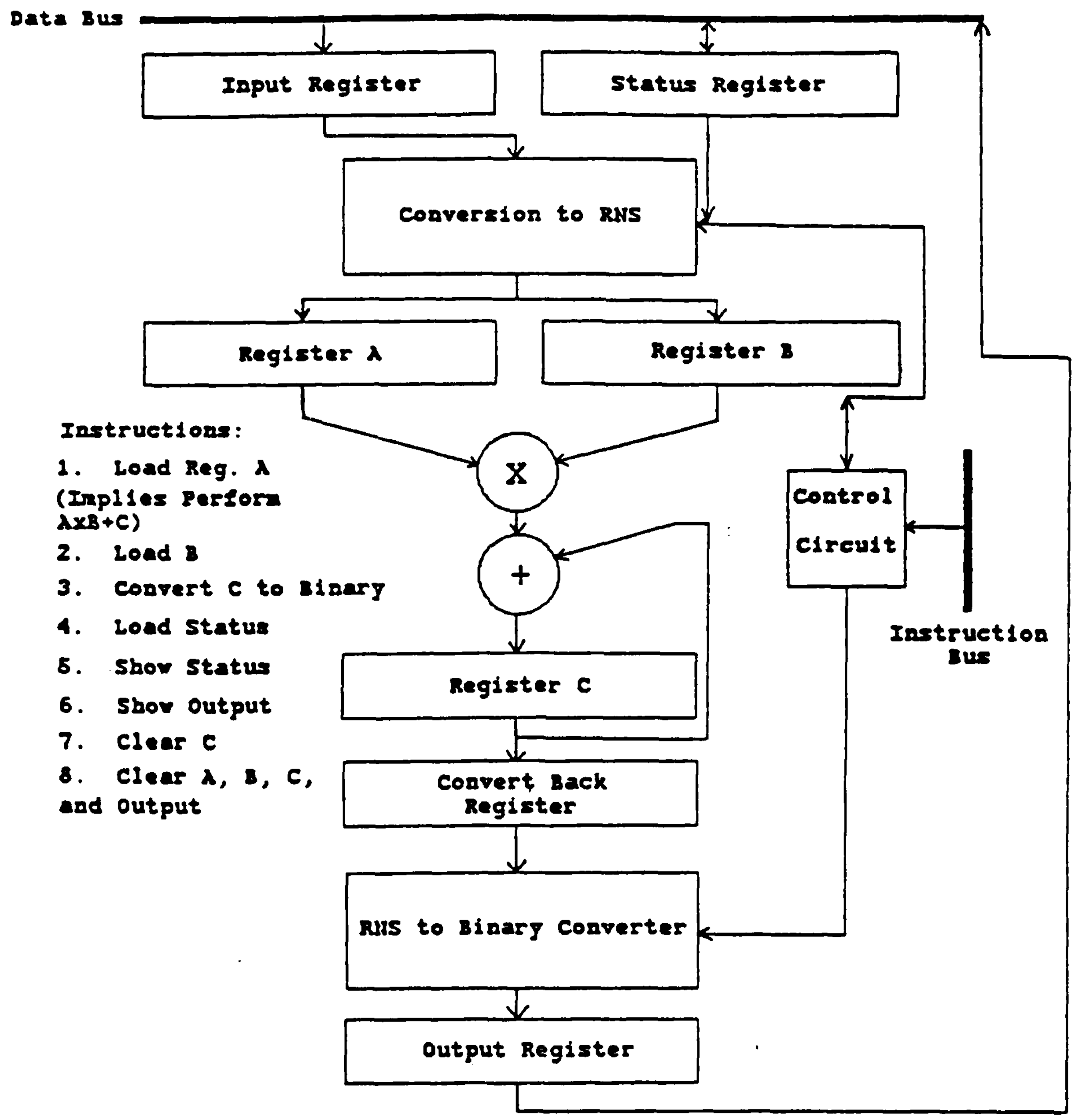


Figure 4.4

Block Diagram of the PRVP

4.2.6 Multiply - Accumulate Section

Although the following discussion is specific for the PRVP, many points and procedures can be generalized for any desired AU.

The use of hybrid adders in RNS processors has already been discussed in some detail in the preceeding sections. The same idea would apply to a residue subtracter; however, the designer should use great care to implement the PLA in the correct way for the chosen RNS system. It may be far easier to input the relevant operands as negative numbers and add them using a hybrid adder rather than using a hybrid subtracter. This would work well in a filter application.

Due to the nature of the conventional parallel multiplier, a hybrid approach for an RNS multiplier cell would not make sense. It would be far faster to use a look-up approach to the problem since the major reason to going to an RNS is to speed up integer multiplication, a notoriously slow operation in a binary processor.

Since modularity of design is important in VLSI, the look-up approach was taken for the AU section rather than the hybrid approach. We know before the design was very far that the RNS adders and multipliers would be very close together spatially in the chip. It is far easier to efficiently use the available space for the AU unit if each of the subcells were the same shape and roughly the same size. Thus, the choice of a look-up for the adder portion of the AU was justified.

There was a seemingly unsurmountable problem with this approach, however. The 5-bit moduli required the use of 10 input PLA's with 1024 minterms required. The resulting PLA's from this direct approach were approximately 10 times what could fit onto the chip. A unique heuristic solution to this problem was found, and was then refined even farther.

We could use the approach of Fig. 4.5. This seemed like a very good idea on paper, but had a major problem which will be discussed shortly. The approach of Fig. 4.5 is a simple one conceptually and will be explained by a sequence of examples.

For every input term, we could eliminate from the input, we could cut the resulting PLA size approximately by half. This is due to the number of minterms being divided by two. Recall that the height of the PLA is directly proportional to the number of minterms it implements. Therefore, if we remove one of the inputs from the input field, say the least significant one, then we could use this bit as the select line to a 2:1 multiplexer and select between two PLA's. One PLA would implement the odd terms, and the other, the even terms. Therefore, if each of the PLA's shared the same inputs, two answers would be generated as if the least significant bit were even or odd. The 2:1 multiplexer would then choose the correct answer based upon what the least significant bit was. Therefore, we have cut the huge PLA into two different pieces with an additional cost of several 2:1 multiplexers (one for each output bit). We could extend this idea for an arbitrary number of inputs (within reason) eliminated from the PLA input field. It would be a good idea to match the delays through the PLA stage and the multiplexer stage to find a optimal combination.

What has this gained? In theory, the fragmented version of the look up should be the same size as the original, unfragmented version of the PLA. What saves us from this is the ESPRESSO program. It turns out that the smaller the PLA, the better ESPRESSO is able to minimize the number of required minterms. Therefore, we can usually expect an area savings by using this method. Another important advantage to this method is that we have essentially broken up the huge, slow single PLA into smaller, parallel faster ones. Thus we have achieved a very rare combination in VLSI design–we have both decreased the size of a logic function and increased its speed simultaneously.

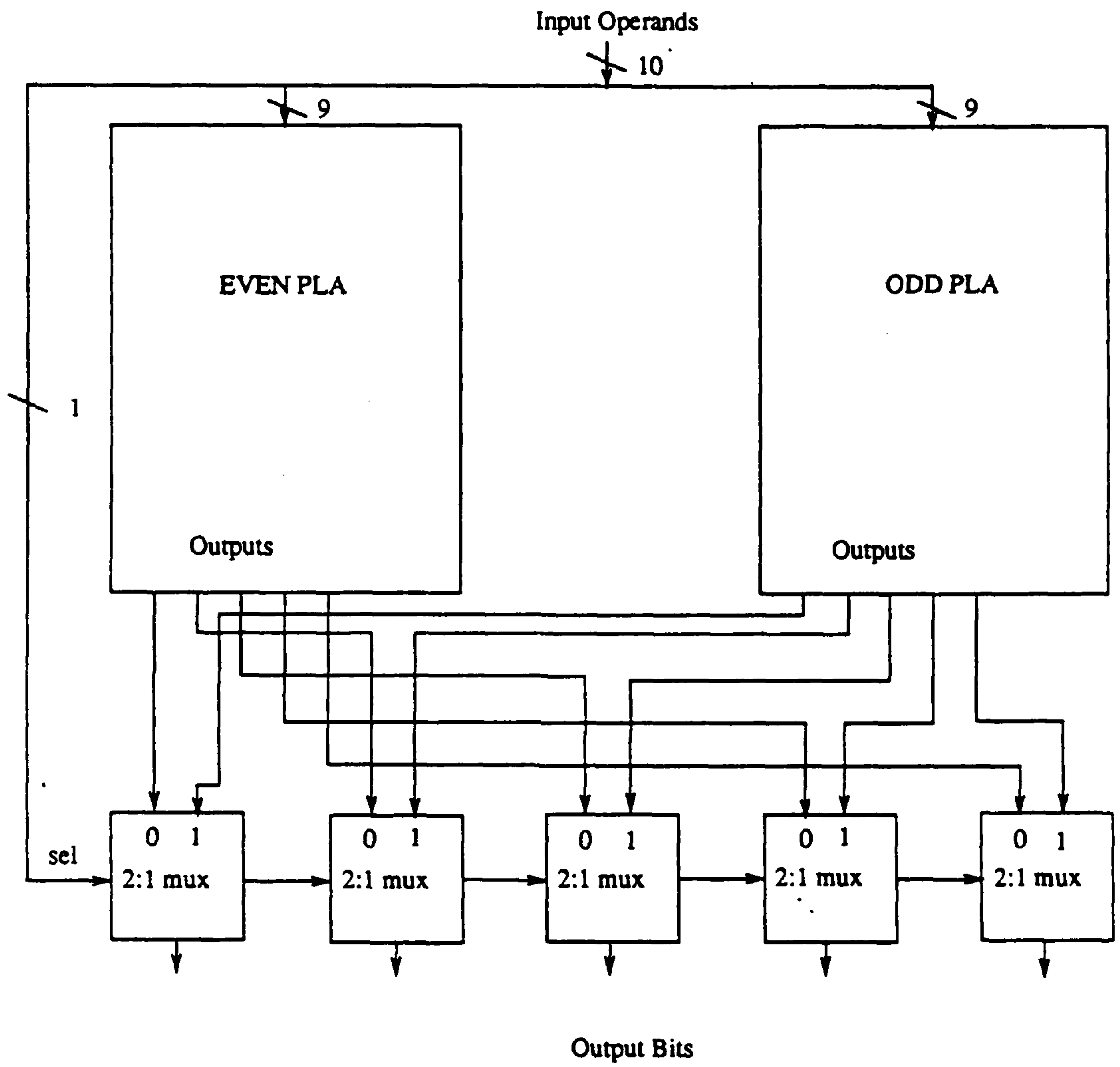


Figure 4.5

PLA Size Reduction

The main problem of this approach is that we have neglected the routing from the outputs of the PLA's to the multiplexers. Fig. 4.6 illustrates this point. There are so many signals that must cross each other that the interconnect area eliminates the size advantage of this approach. This was learned after every PLA in the PRVP was generated and was in the file system.

As compared to inputs, additional outputs do not increase the size of PLA's nearly as much as inputs. After some time, we found a solution to the problem of Fig. 4.6 . A mapping similar to the one just discussed could be made, but the job could be portioned in a different way. We could assign one PLA to each bit of the resulting output field. For example, a modulo–31 adder would have five bit outputs, so we would assign five PLA's for the implementation of the function. We could then "peel off" some of the inputs as before, say the two least significant bits from one operand and the least significant bit off of the other operand. Each of the PLA's would be programmed to yield eight different answers for a given 7–bit input. These eight outputs would correspond to the eight different correct answers as if the three bits that have been eliminated from the input space of the PLA were actually in the input field. We then use the three bits as select lines for an 8:1 multiplexer which would select the correct answer in all five PLA's. Fig. 4.7 illustrates how the interconnect problem was vastly reduced. This method was smaller than the method of Fig. 4.6 since the number of PLA's was reduced from 8 to 5. For smaller width moduli (4 to 6 bits), this method works ideally as compared to the method of Fig. 4.6.

We could arbitrarily choose the bits from the input field that go to the multiplexer. However, we could take advantage of the restricted input field magnitude to help ESPRESSO reduce the PLA size. For example, we know that if we are constructing a modulo–17 adder, the inputs will not have a magnitude of over 16. So, if we use the least significant bits of each input operand, we can take advantage of all the DON'T CARE conditions of the inputs to minimize the outputs. This is the approach taken in the AU of the PRVP.

Fig. 4.8 illustrates a typical layout of the AU section in the PRVP. All the other moduli have exactly similar arrangements. It is evident that the connections from the PLA to the MUX for each output bit are very compact and contain no crossovers, which greatly eased the layout. The adder and the multiplier were laid out separately and assembled together later to help automate the design process. Obviously, the use of a "template" would help construct of all these units. A template is a master cell with the PLA's removed but with markers inside the cell where each of the PLA's would go. The designer merely drops each of the desired PLA's at its marker and makes small corrections to the design depending on the actual size of each PLA. This method reduced the construction of the AU components from 90 minutes (starting from scratch each time) to an average of 15 minutes per unit. When there are more than 100 units to construct (including the convert back hardware), the time savings is enormous. Of course, automating the design process at this stage would be indicated from the modularity of each cell in a general AU. Those with excellent software ski'' could automate 95% of an RNS processor.

We could have used the portioned method of PLA construction in the conversion from binary to RNS. However, the shape of the resulting cell was not right for the floorplan of the PRVP. Thus, straight look–ups were used instead of the partitioned PLA's.

The Microsoft QuickBASIC programs used to generate the cells in the AU of the PRVP are given in [36].

4.2.7 Mixed Radix Conversion

The conversion of a number in RNS format to binary using the Chinese Remainder Theorem involves binary multiplication in its intermediate steps. Operations such as

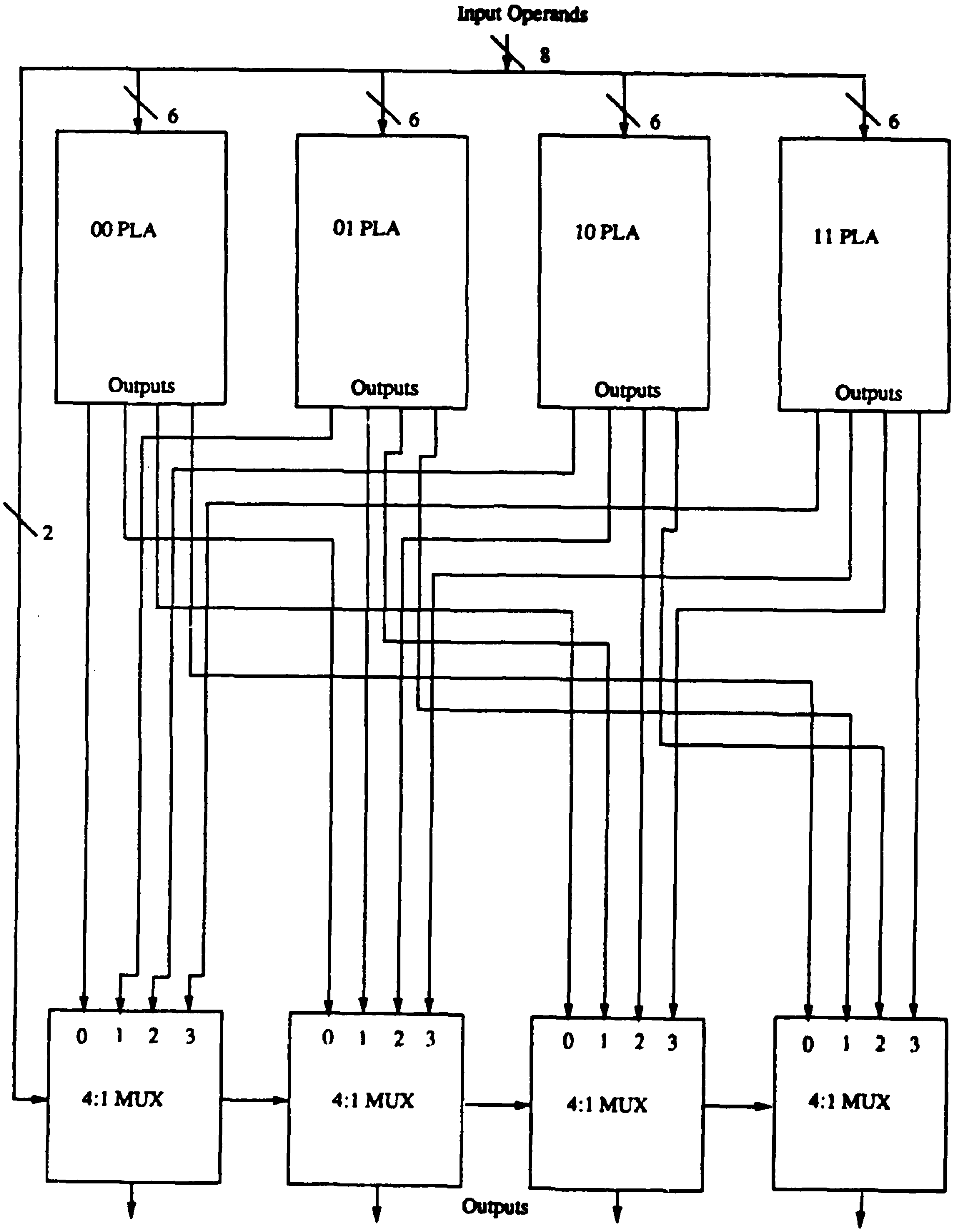


Figure 4.6

Cost of Interconnect in PLA Size Reduction

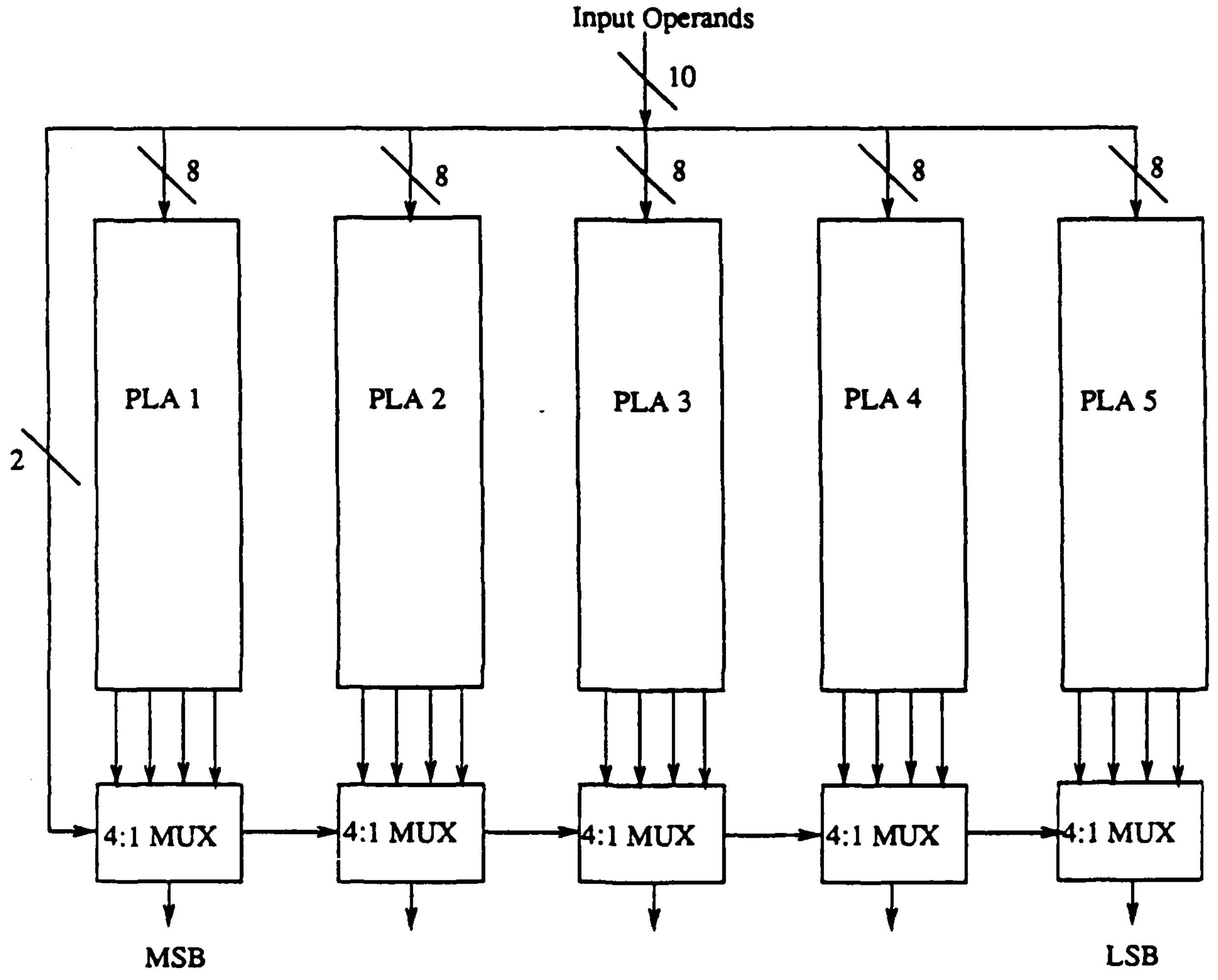


Figure 4.7

Solution to the Interconnect Problem

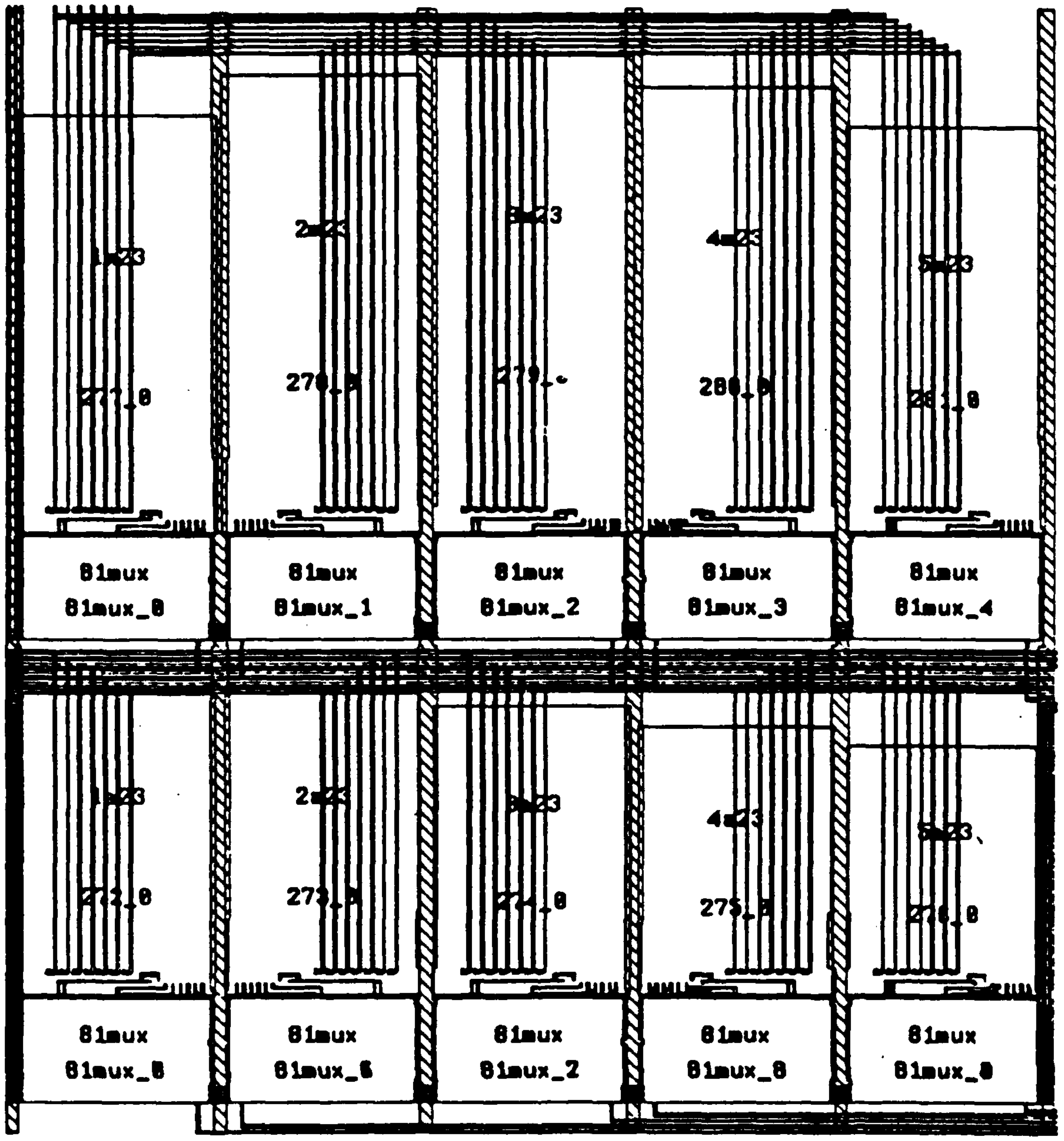


Figure 4.8

Typical AU Piece in the PRVP

binary multiplication are what an RNS was trying to avoid in the first place. This is why the mixed-conversion process using the Modified Chinese Remainder Theorem is preferred due to the use of modular arithmetic in the intermediate stages. The conversion of a number from RNS to unsigned binary is given by the following equation [21]:

$$x = a_n \prod_{i=1}^{i=n-1} m_i + \cdots + a_3 m_2 m_1 + a_2 m_1 + a_1$$

where x is a mixed radix number, the m_i's are the moduli in the system, and the a_i's are coefficients to be found by the RNS to binary converter. This equation is first taken modulo m_1. Then, all of the terms except for the last a are divisible by m_1 which means that they all are identically zero. Therefore, $(x)\mathrm{mod}(m_1) = a_1$ which implies that a_1 is the first residue digit. Since $(x-a_1)/m_1 = b_1$ is an integer value without a remainder, $(b_1)\mathrm{mod}(m_1)$ exists and is equal to a_2. Repeating this process for all the moduli in the system yields all the required coefficients.

We can then get the coefficients by a residue subtraction and residue division applied successively. The division can be reduced to multiplying by the multiplicative inverse of the moduli we are working on. The multiplicative inverse of a number n modulo m is the solution to the following:

$$(ni)\mathrm{mod}(m) = 1$$

where i is the number that is desired. The number i always exists for all n's as long as m is a prime number.

This process is illustrated with the following example 4 moduli system:

$$(8,5,7,3)$$

Assume that the residue representation of a number is (7,4,6,2). Starting from left to right, we see that 7 is a_1 and we then subtract 7 from the three remaining digits. This leaves (x,2,6,1) which we multiply by (x,2,1,2) which is the multiplicative inverse of 8 of the remaining moduli. We then have (x,4,6,2), and 4 is the value of 4 is subtracted from the remaining moduli which yields (x,x,2,1) which is multiplied b· (x,x,3,2) to yield (x,x,6,2). 6 is the value of and is subtracted from the remaining moduli to yield (x,x,x,2) which is then multiplied by (x,x,x,1). Therefore 2 is the final coefficient and the number is:

$$x = 7 + 4(8) + 6(8x5) + 2(8x5x7) = 839$$

This process only gives the positive (unsymmetric) version of x. If a symmetric system is used, the binary number 839 would be added to (the negative of the dynamic range) to get the correct signed result.

Fig. 4.9 shows the hardware implementation of the conversion from RNS to binary for this example. The subtraction and multiplicative inverse operations can be performed by one PLA. The coefficients are fed to a PLA which multiplies by the required radix, and is then summed up by binary adders. Fig. 4.10 shows the required hardware which detects and converts the output of the Mixed-Radix Conversion (MRC) process to a signed 2's complement output.

Reference [36] contains the Microsoft QuickBASIC code which generates all the PLA's for a MRC converter. The Radix PLA's (the PLA's which multiply the coefficients by the radix) were generated by hand using the binary calculator tool available on the SUN workstation.

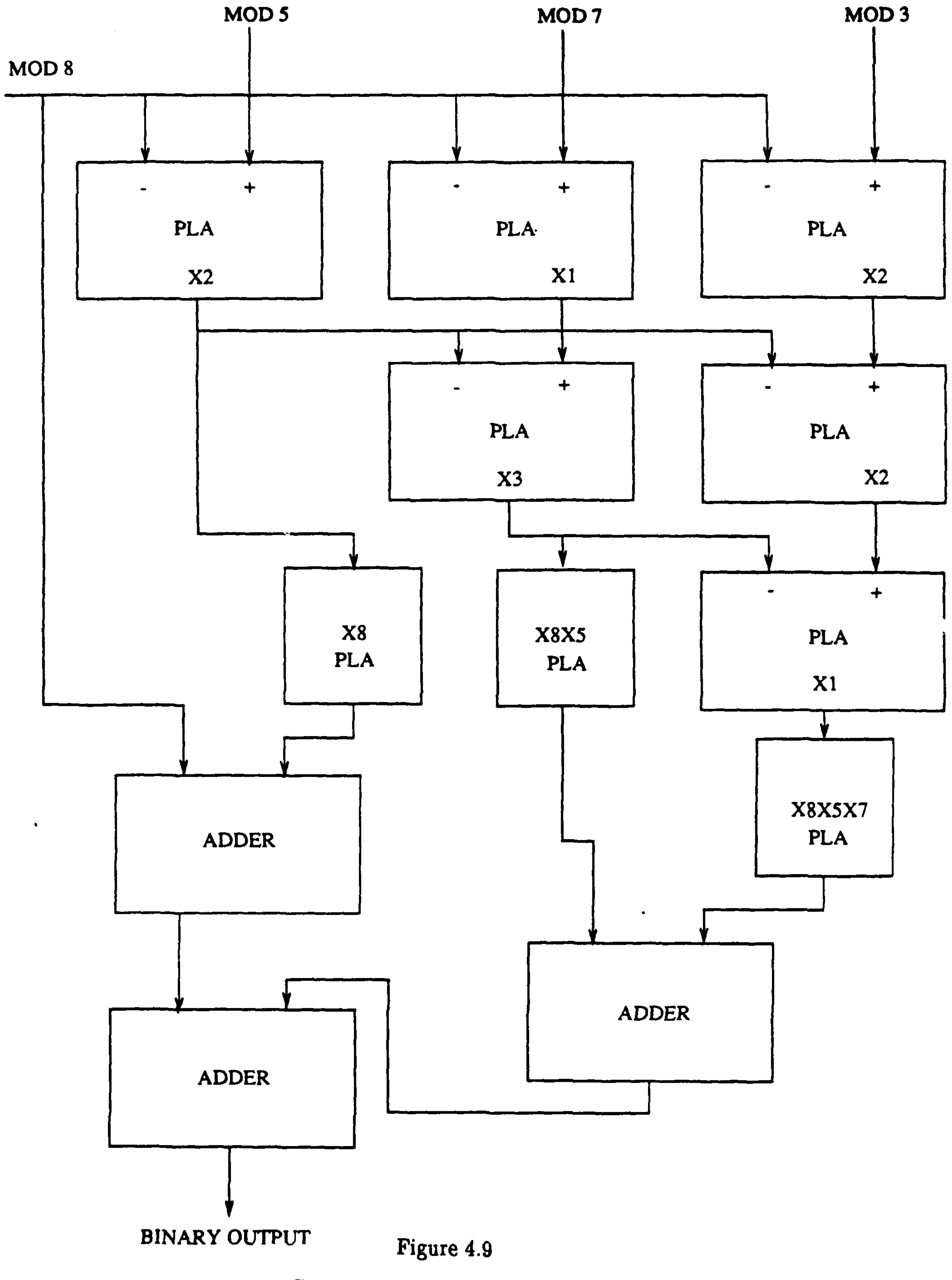


Figure 4.9

Conversion from RNS to Binary

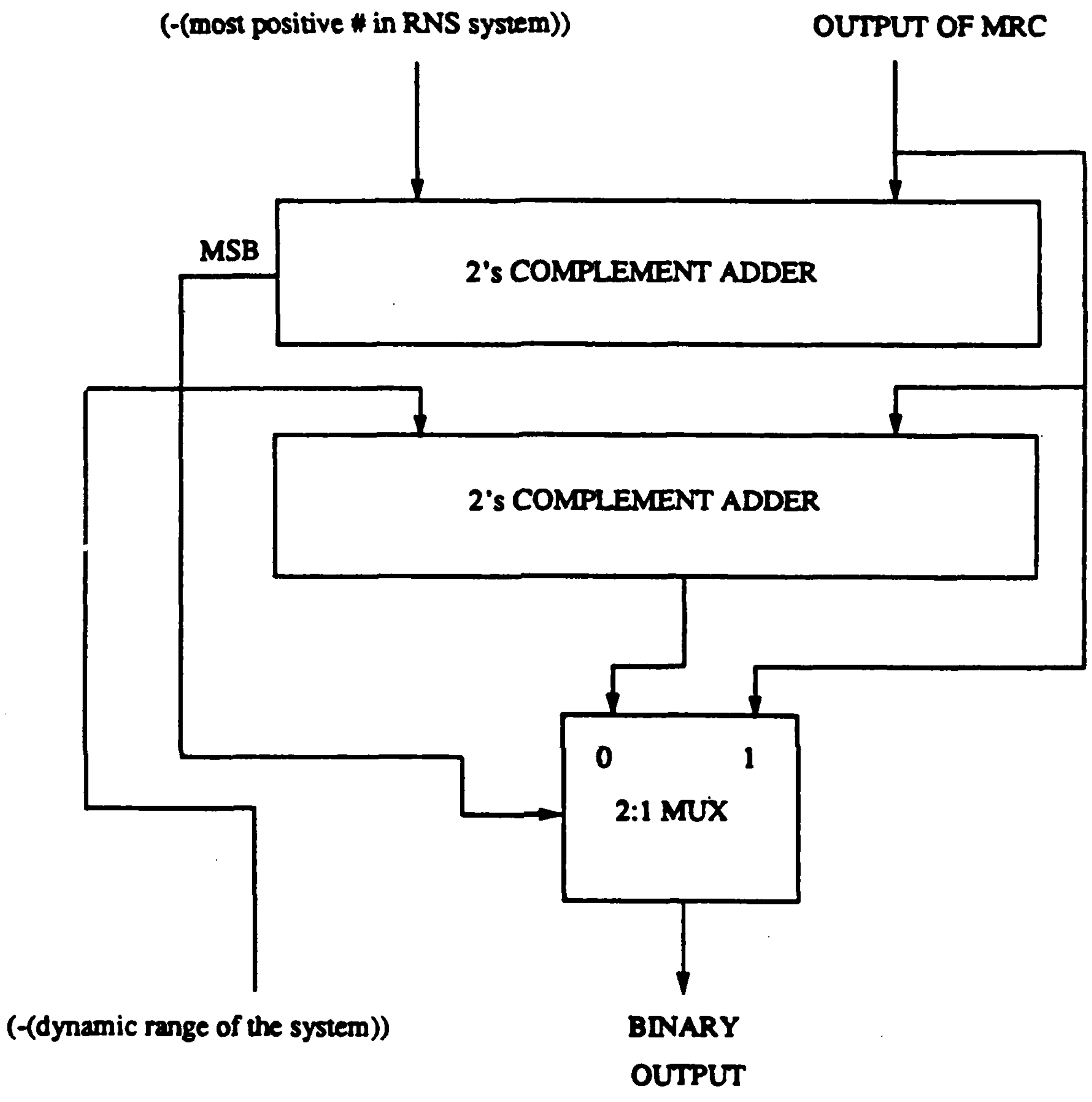


Figure 4.10

Sign Detection Hardware

The binary adders at the end of the MRC can be implemented using carry-select adders to increase the speed of the conversion process. This is the method used in the PRVP. Notice that the size of the adders depends on the range of the inputs to them. Therefore, the adders near the top of the process which adds the numbers resulting from the coefficients generated first are not as wide as those near the bottom generated from the later coefficients.

Fig. 4.11 shows the MRC in the PRVP. This is a large structure, and contains many interconnects. Notice that some of the PLA's were not partitioned as described previously. This was done so that the width of the MRC would fit into the over-sized frame used for the PRVP. If the block diagram of Fig. 4.9 were extended for an 8 moduli system, it would follow Fig. 4.11 exactly. However, the sign detection circuit of Fig. 4.10 was modified in order to increase the speed of the MRC.

We could eliminate the first block of Fig. 4.10 which detects the magnitude of the number coming out of the MRC if we partitioned the RNS so that we could tell the sign of the output of the MRC by looking at the first bit. In other words, if all numbers from 00000000 to 7FFFFFFF were positive and 80000000 to FFFFFFFF were negative, we could detect the sign of the number by looking at the MSB. However, since the dynamic range of the PRVP's RNS system is larger than 32 bits, possible errors may result at first glance if we chose the system [2147483647,–4537866023] for the (31x29x23x19x17x13x11x7) dynamic range of the system. In other words, we map all numbers with a zero in the MSB as positive numbers and all others as negative numbers in the system. All of the numbers greater than 4294967296 are mapped out of harm's way into the negative numbers less than –2147483648 which exceeds the bus size of 32 bit 2's complement numbers. Therefore, if the system is partitioned in this manner, the MSB and the carry out of the final carry-select adder (bits 32 and 33) could simply be OR'ed together to detect the sign of the number. This sign information along with the desired format (does the programmer want true or 2's complement outputs?) stored as a single bit in a status register could control the 2:1 multiplexers at the bottom of the MRC. This is the system partition chosen for the PRVP. The constant added to the bottom if the sign is negative is simply the 2's complement form of (–6685349671), the dynamic range of the system.

4.3 PRVP DESIGN DETAILS

4.3.1 General Considerations

The designer of an RNS processor will find that most effort will go into floorplanning and routing than is usually devoted for a fully custom design. It is easy to see this since nearly all of the basic cells in an RNS processor is generated automatically by the use of software tools such as MPLA, which was discussed earlier. The design of an RNS processor is almost as fast as a semi-custom processor due to this fact. The difference lies in optimizing the floorplan and interconnect by human rather than machine intelligence.

It could be stated that the inputs and outputs of an RNS functional block are grouped in specific areas of the cell in question. These blocks are usually combinational in nature; therefore, there are only data signals to route with no sequential control signals to worry about. All control signals are attached to internal registers and pipes, which are composed of either dynamic or static elements.

As a result, a register-transfer architecture is usually used in an RNS processor. This design could be pipelined to nearly any depth, since an RNS processor will be used in DSP (Digital Signal Processing) where latency isn't generally any problem. However, if the

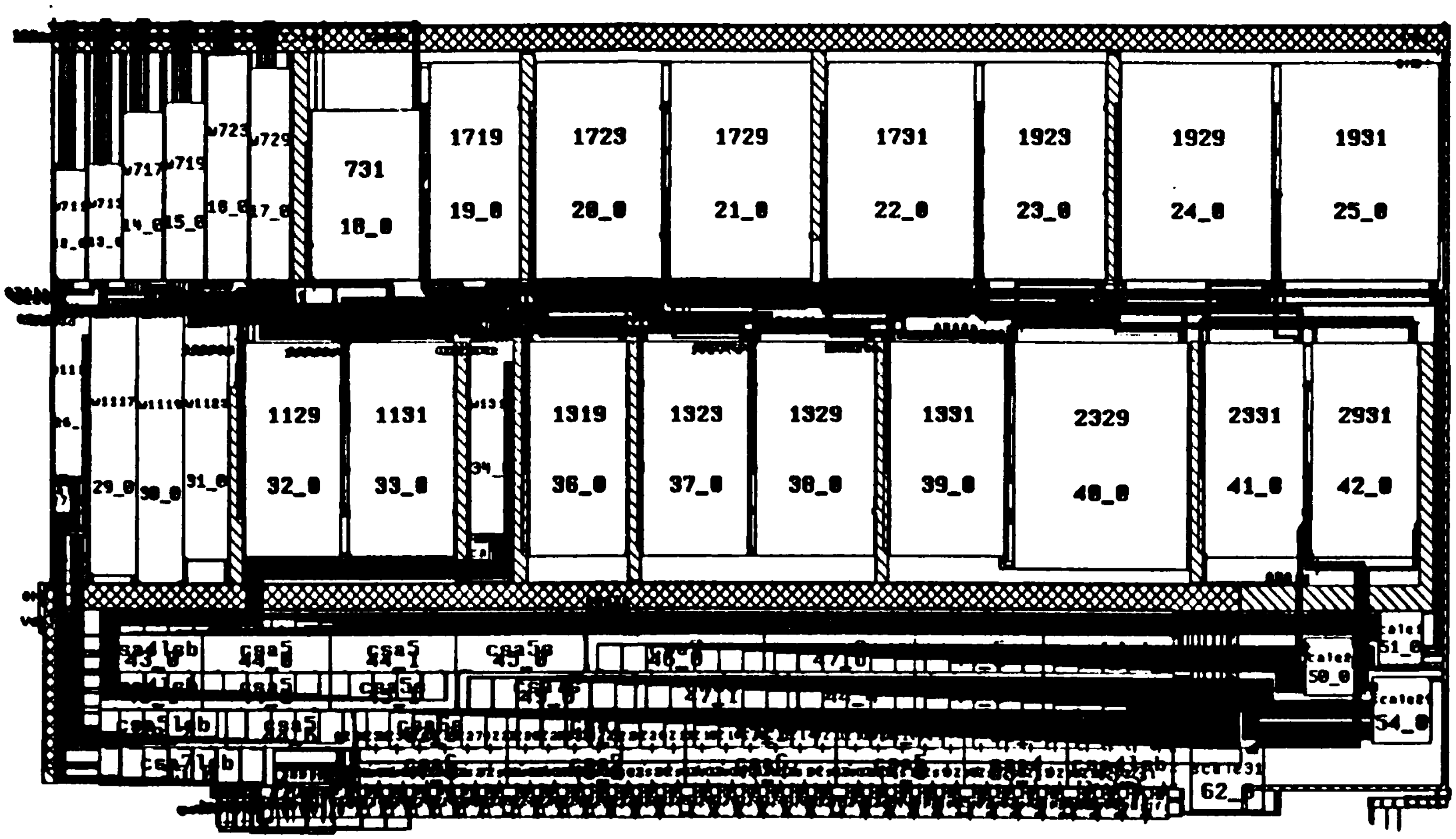


Figure 4.11

Lay Out of the Conversion to Binary Hardware

RNS processor has a nontrivial instruction set, dependencies may occur which must be accounted for either in the hardware or in the compiler design of the system if pipelining is used.

It can then be stated that the RNS portion of the processor is driven by the functionality and the placement of registers and pipes is driven by the timing (speed) specifications of the processor. This is only true in an upper-level design-sense, since both these areas have some overlap. Controllers for the internal registers can be designed using some standard state-machine design techniques.

The pieces of a RNS processor can be reused in other designs. Extending this idea, an engineer can fabricate conversion to RNS and RNS to binary chip sets with the AU portion of the processor specifically designed. Extending this idea even further, a library of RNS cells can be easily constructed which could be used by existing auto-routing and auto-placement software to build semi-custom chips. The design of such a software system is one logical extension of the work presented in this section.

4.3.2 Details of PRVP

Nearly all control systems can be expressed in matrix format. This means that the implementation of these processors uses a vector-processor approach to the architecture since calculations involving matrices can be decomposed into vector operations. One useful element for a vector processor is a multiply-accumulator. This is the basic function decided upon for the PRVP. At first, the operands were specified to be 32 bits wide. However, this made little sense from a signal processing standpoint since most available Analog to Digital Converters (ADC's) are 8 bits wide with 16 bits being the larger size available. These are not currently available in speeds approaching the PRVP, so the PRVP can handle the calculations for several different control systems with data coming from several ADC's which may be polled by the SF1 master processor. Since the PRVP was a fat design to begin with, the conversion from binary to residue for 32 bits was trimmed to 16 bits. Also, 16-bit operands yield a 32-bit result when they are multiplied together. If 32-bit operands were used with a 32 bit dynamic range, the programmer would have to worry constantly about overflow.

The projected maximum speed for the SF1 stack bus clock is 10 Mhz. Since operations are performed on every phase of the clock, pipe delay times of 50 ns are required.

The conversion from residue to binary would only be required occasionally as compared to the multiply-accumulate operation, so the conversion from binary to residue and the AU was optimized for speed. The converter for residue to binary was optimized for speed as area allowed. In a controls environment, signed operations are required, so the PRVP was designed to handle signed and unsigned operands, with the choice being up to the software developer.

The controller in the PRVP is specific to the SF1 stack bus. However, it would take about 2 days to change the control circuit for a different control scheme. This illustrates how independent an RNS processor is to the way it receives instructions due to the register transfer architecture. Other processors would have to be completely redesigned to effect a control scheme change, since they have control signals embedded into their logic.

Figure 4.12 shows the entire layout of the PRVP. The conversion hardware dominates the area of the chip, with the AU, the control circuit, and the internal registers taking up a small part of the available area. The size of the design is roughly 10,300 microns square with over 130,000 transistors inside it. It is the largest chip designed at WSU to date. Fig. 4.13 shows a floorplan of Fig. 4.12 so that each section of the design can be seen more clearly. It can be seen that interconnect has been painstakingly optimized to reduce the area required by the design.

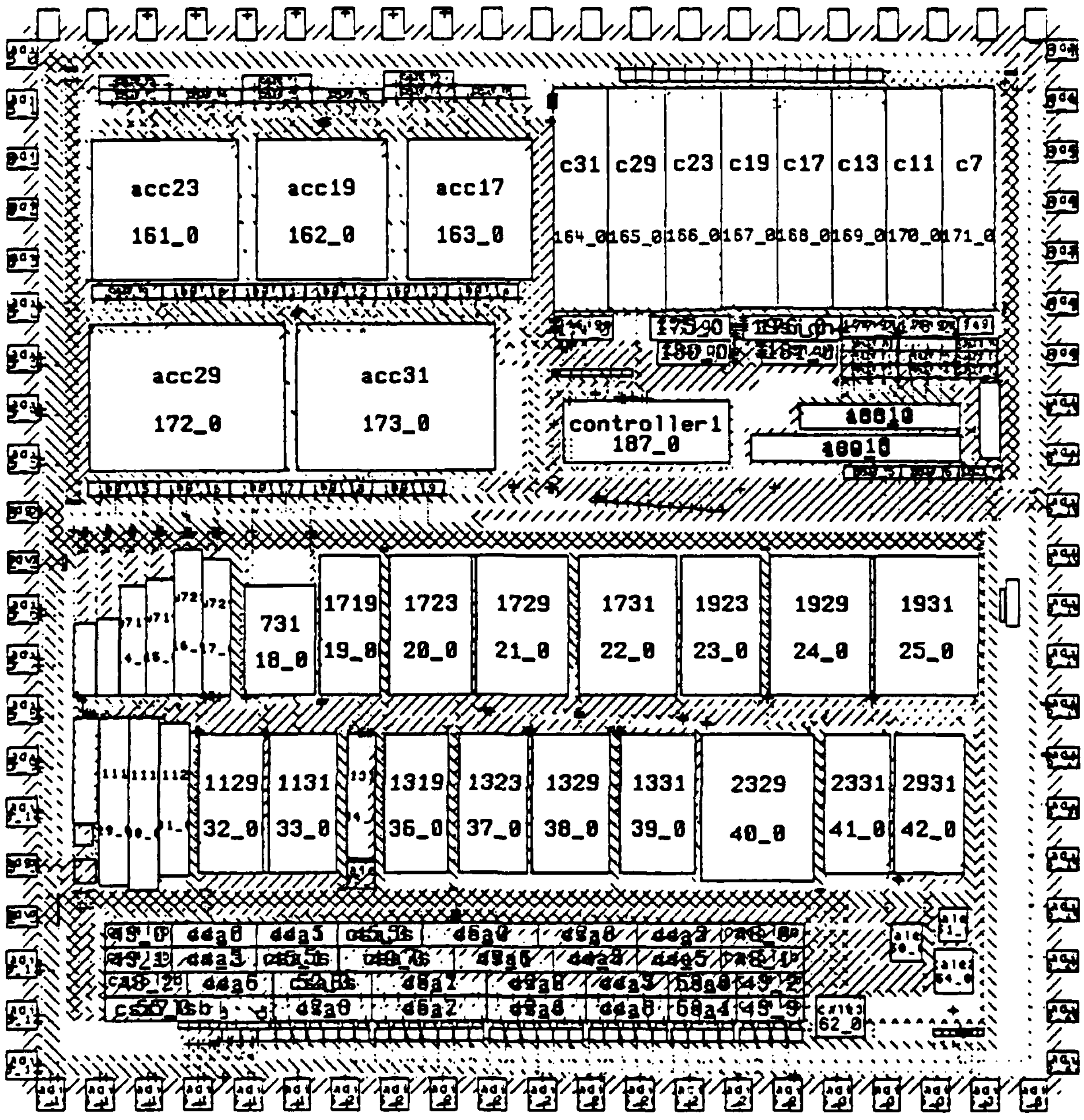


Figure 4.12

Lay Out of the PRVP

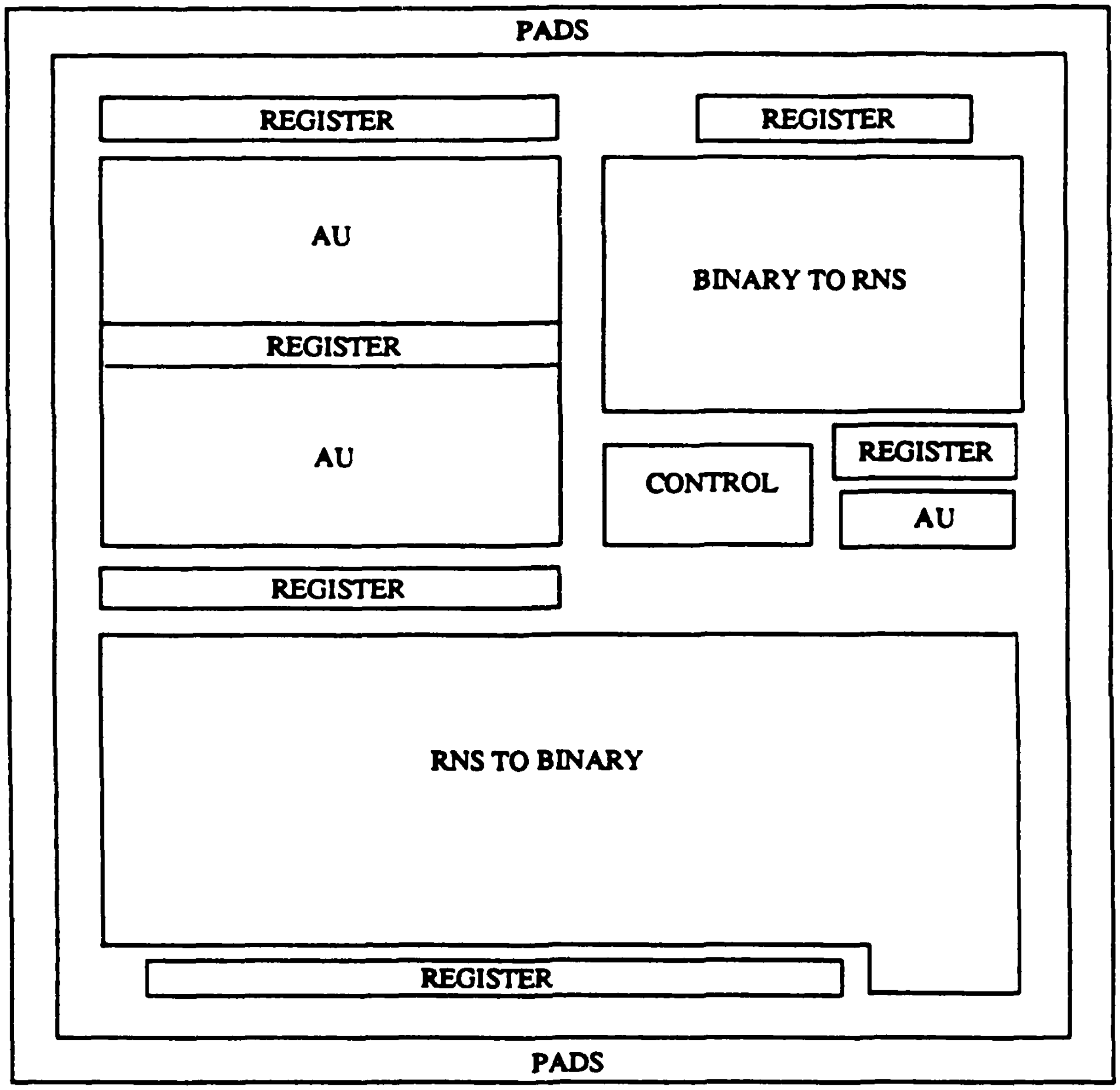


Figure 4.13

Floor Plan of the PRVP

4.3.3 Conversion From Residue to Binary Details

A block diagram of the conversion from binary to residue is shown in Fig. 4.14. The structure is composed of the conversion PLA's which maps two 8 bit fields of the input into their particular residue representation. The output of these PLA's are added by a ripple–carry adder which is composed of 2–bit binary full adder circuits. The layout of the full adder circuit is shown in Fig. 4.15, which is a modified version of a full adder circuit available in the standard cell library. The output of the adder circuit is connected to a PLA which converts the output of the ripple–carry adder into the correct residue representation with respect of the particular moduli. This PLA contains an additional bit in the input field which, if set, corrects the output of the adder and converts the number from 2's complement format (if the input is negative) to the correct negative residue representation. This bit is derived from the sign bit from the status register (set if the programmer wants negative numbers) and the MSB of the binary input (set if the number is negative). These two bits are simply ANDed together to obtain the control signal.

As stated earlier, a copy of Fig. 4.14 must be constructed for each moduli in the system. In the PRVP, the slowest block of this sort is the mod 29 block. The delay through the conversion PLA's was simulated to be 18 ns, the delay through the adder block was simulated to be 10 ns, and the delay through the final PLA was simulated to be 8 ns. The total delay is 36 ns simulated, using the SPLICE simulator for each test. This simulator is roughly as accurate as SPICE level 1 modeling.

4.3.4 AU Details

The AU is composed of residue adders and multipliers. Included in the AU section are the A, B, and C registers. Fig. 4.16 shows a block diagram of the AU section of the PRVP.

The AU section is actually concentrated in two areas of the design. The 5–bit moduli are grouped together in the upper left corner of the design, and the 4– and 3–bit moduli are located on the center right part of the design.

The largest (slowest) cell in the AU is the mod 29 AU cell. The combined adder and multiply delays are 37 ns simulated using SPLICE.

Since this section is distributed throughout the design, additional plots would convey no additional information. Refer to the previous figures, including the floorplan, for an idea of how this structure was constructed.

4.3.5 RNS to Binary Converter Details

The lower half of the design is mostly devoted to the RNS to binary converter. This structure is composed of PLA's and carry–select adders. The MRC scheme outlined previously is followed in this cell.

The carry–select adders were constructed from the standard cell library full–adder circuit used previously. A block diagram of this circuit is shown in Fig. 4.17. The advantage of this approach over a ripple–carry adder is the decreased delay time gained at the expense of increased area used to implement the adder. The delay through this cell depends on the width of the adder, and adders from 7 to 32 bits wide are used in the converter. The delay through the 32–bit carry–select adder was simulated to be 23 ns. A

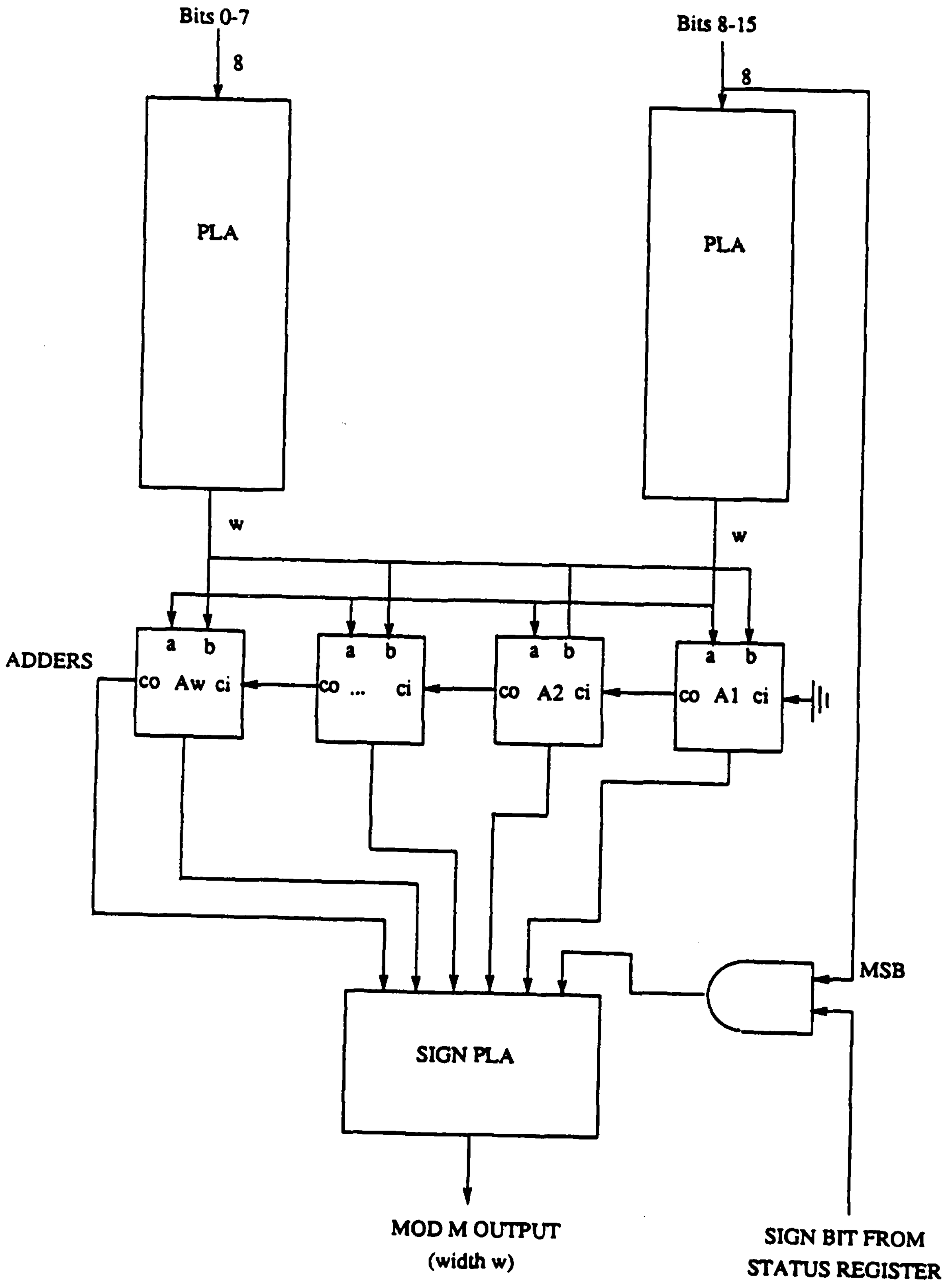


Figure 4.14

Detailed Schematic of the Binary to RNS Hardware

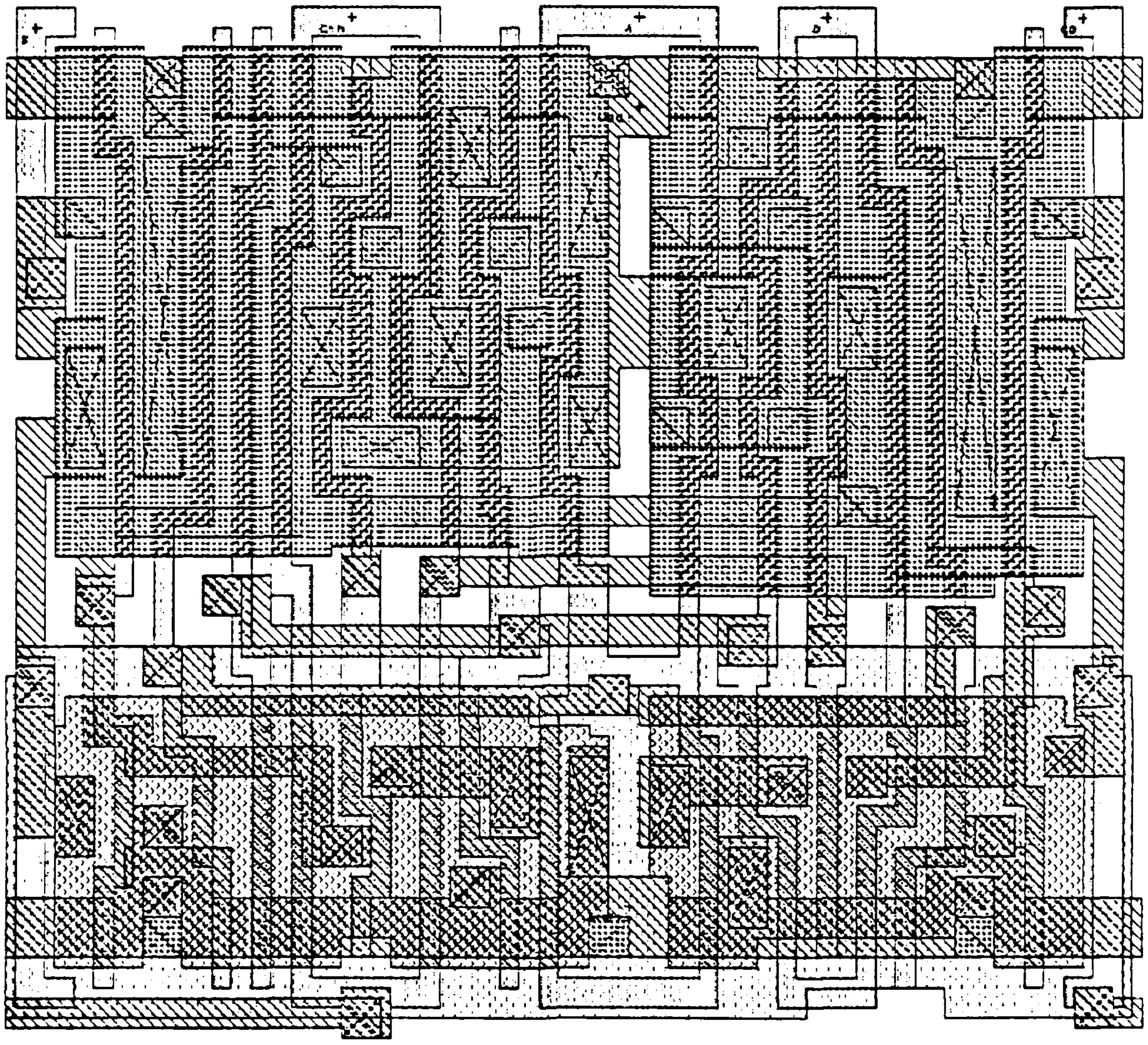


Figure 4.15

Lay Out of the Full Adder

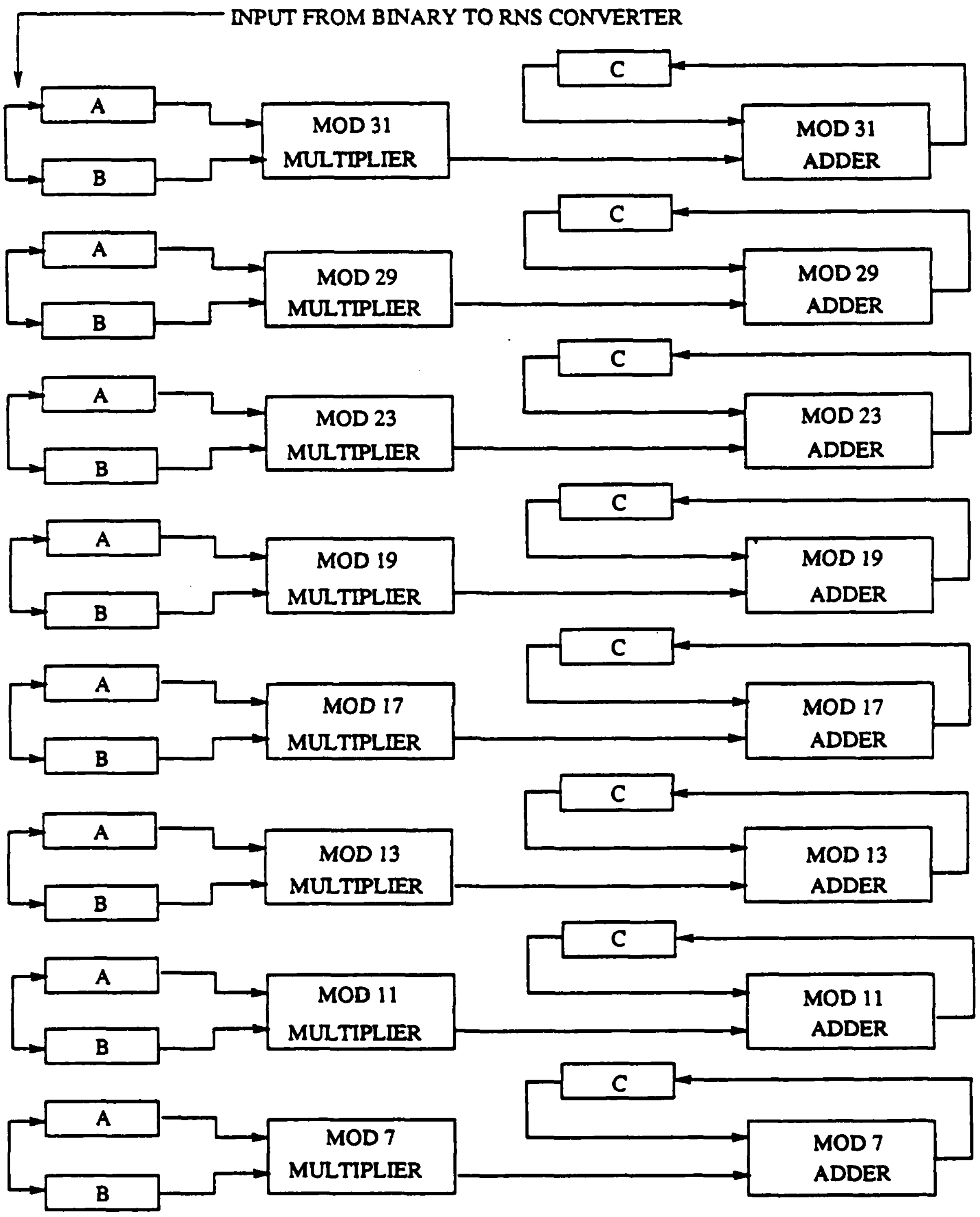
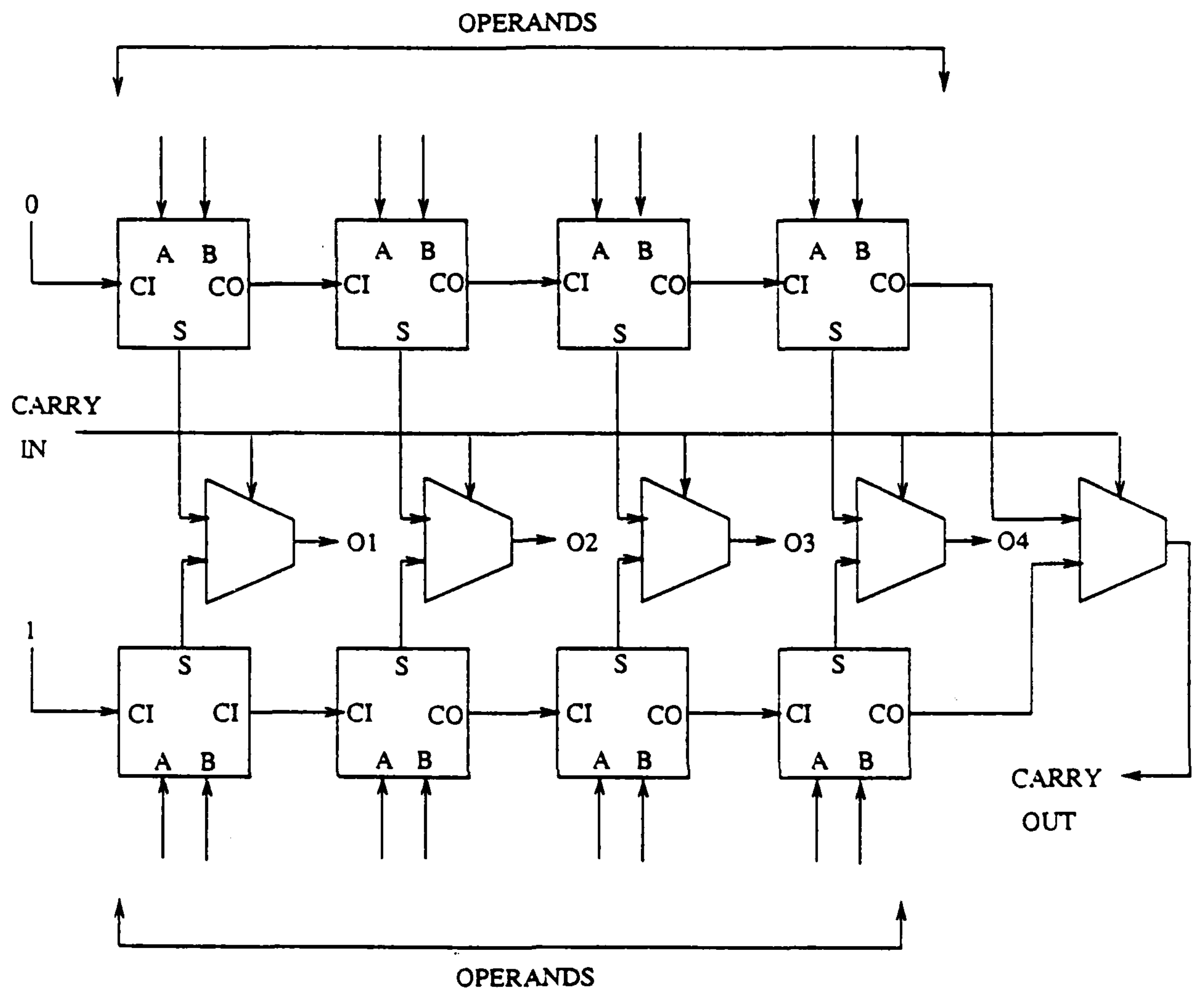


Figure 4.16

Block Diagram of the AU



O1-O4 are the sum of the two operands.

Figure 4.17

Block Diagram of the Carry Select Adder

ripple–carry adder of the same width using the same full–adder cells would have a delay of over 70 ns. Clearly, the increased area used for the carry–select adder was worth the decrease in delay.

The estimated delay of the MRC processor cell is 260 ns., based upon SPLICE simulation of the smaller cells it is composed of.

4.3.6 Controller and Interface

The PRVP was designed to interface directly with the SF1's stack bus. This bus's main function is for data transfers between the SF1 and its smart memory chips, called stack frames. These stack frames act as either random access or stacks, depending upon the needs of the particular application. The bus is a simple one, where an address is received by the chips at the beginning of a clock cycle. If the chip selects are asserted during one phase of the clock, data is passed from the SF1 to the memories. If these chip selects are asserted during the other half of the clock period, data is passed from the memories to the SF1. This bus also serves as a bus for peripheral devices, such as the PRVP.

Figure 4.18 shows a timing diagram of the stack bus in action. It is easy to see that micro–coded instructions received as addresses to the PRVP could act as an easy interface to the SF1. This is the method used in the design of the PRVP.

The instruction set of the PRVP is shown in Table 4.2. All of the available address and chip select lines are decoded in the PRVP, in order to obtain maximum flexibility in the address space.

The address and chip select lines are treated as instruction lines by the PRVP. A block diagram of the controller is shown in Fig. 4.19. On every phase of the bus clock, the PRVP latches the address on the address and chip select busses into the instruction registers. This address is decoded by the relevant PLA, and is sent into a sequence of FF's with logic depending on the instruction received. Since the state machine essentially controls when data is latched into the FF's by bringing the select on the 2:1 multiplexer either high (when new data is latched into the device) or low (keeping the same value in the device), the output of these control FF's are fed directly into buffers and then into the select lines of the relevant latches they control. The clear lines to these FF's are connected to the controller, and the clear lines of the FF's inside the controller are cleared by the power–on reset. This way, the internal registers of the PRVP and the FF's in the controller can always be put into a known state at the beginning of a task.

The status register information is also shown in Table 4.3. This register can be set up by writing to it or by setting an external pin (low) and using other pins to hard wire the status register. The sign bit sets signed (1) or unsigned (0) operands, the interrupt enable register disables the maskable interrupt (0) or enables it (1), and the done bit signifies if a number has been converted to binary from the last conversion instruction it received.

The time delay between the reception of a conversion instruction and the generation of either an interrupt or setting the done bit is hard wire controllable by using four bits. This allows many different clock rates for the P2 clock without a overly long wait for a conversion process to be completed. A table containing a diagram of rates and the value of these lines is contained in the test results section.

The hardware for this section of the controller is shown in Fig. 4.20. As can be seen, four different delay times can be controlled by this hardware.

There is one dependency inside the PRVP, which is associated with the conversion to binary instruction. If a LOAD A instruction is given and then a CONVERT TO BINARY instruction is given right afterwards, the value in the C register may not be the desired value. The software engineer must wait if the latest value of the C register is to be converted to binary. Of course, other things could be performed, such as changing the

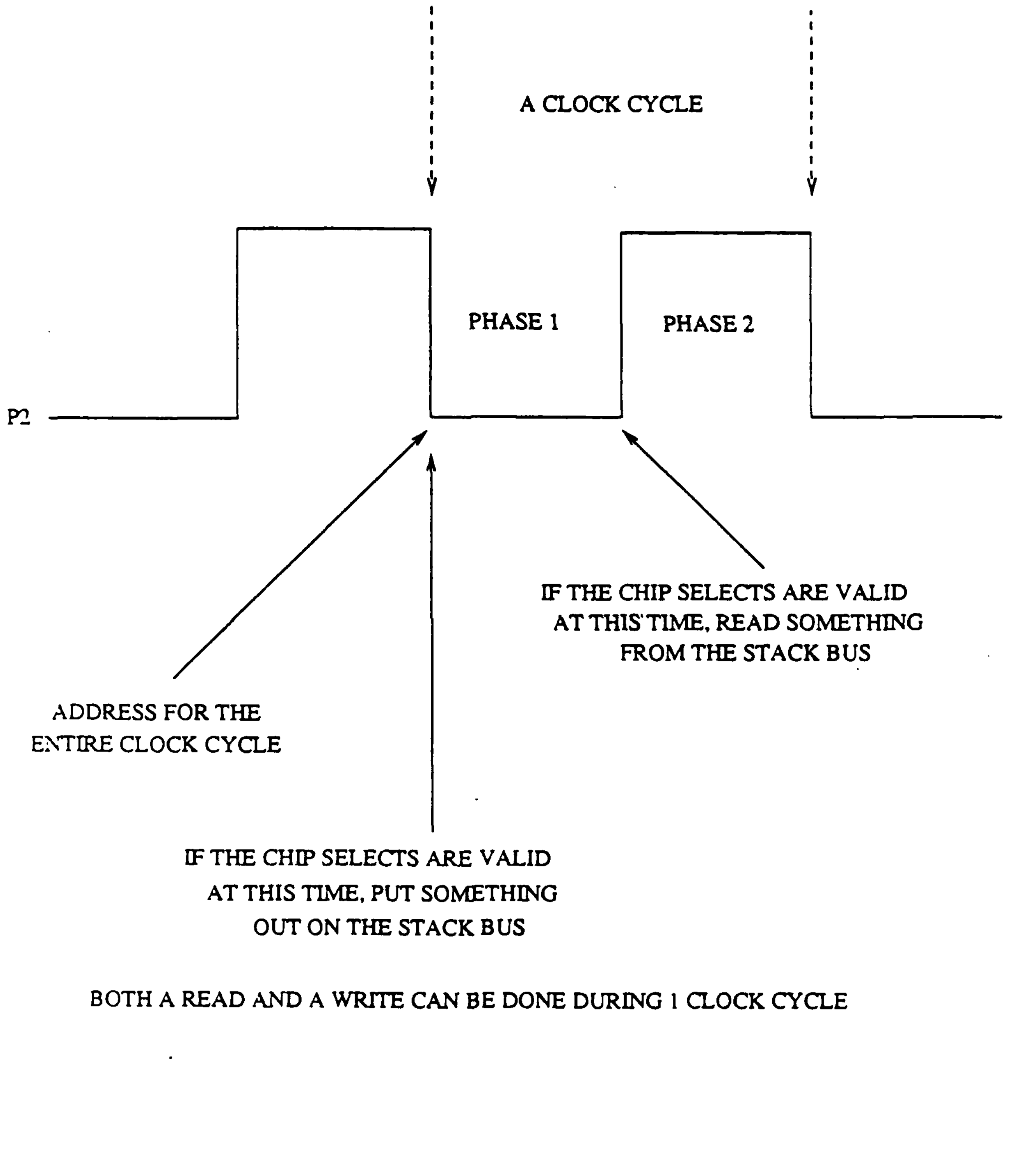

A CLOCK CYCLE
PHASE 1
PHASE 2
P2
IF THE CHIP SELECTS ARE VALID AT THIS TIME, READ SOMETHING FROM THE STACK BUS
ADDRESS FOR THE ENTIRE CLOCK CYCLE
IF THE CHIP SELECTS ARE VALID AT THIS TIME, PUT SOMETHING OUT ON THE STACK BUS
BOTH A READ AND A WRITE CAN BE DONE DURING 1 CLOCK CYCLE

Figure 4.18

Stack Bus Timing

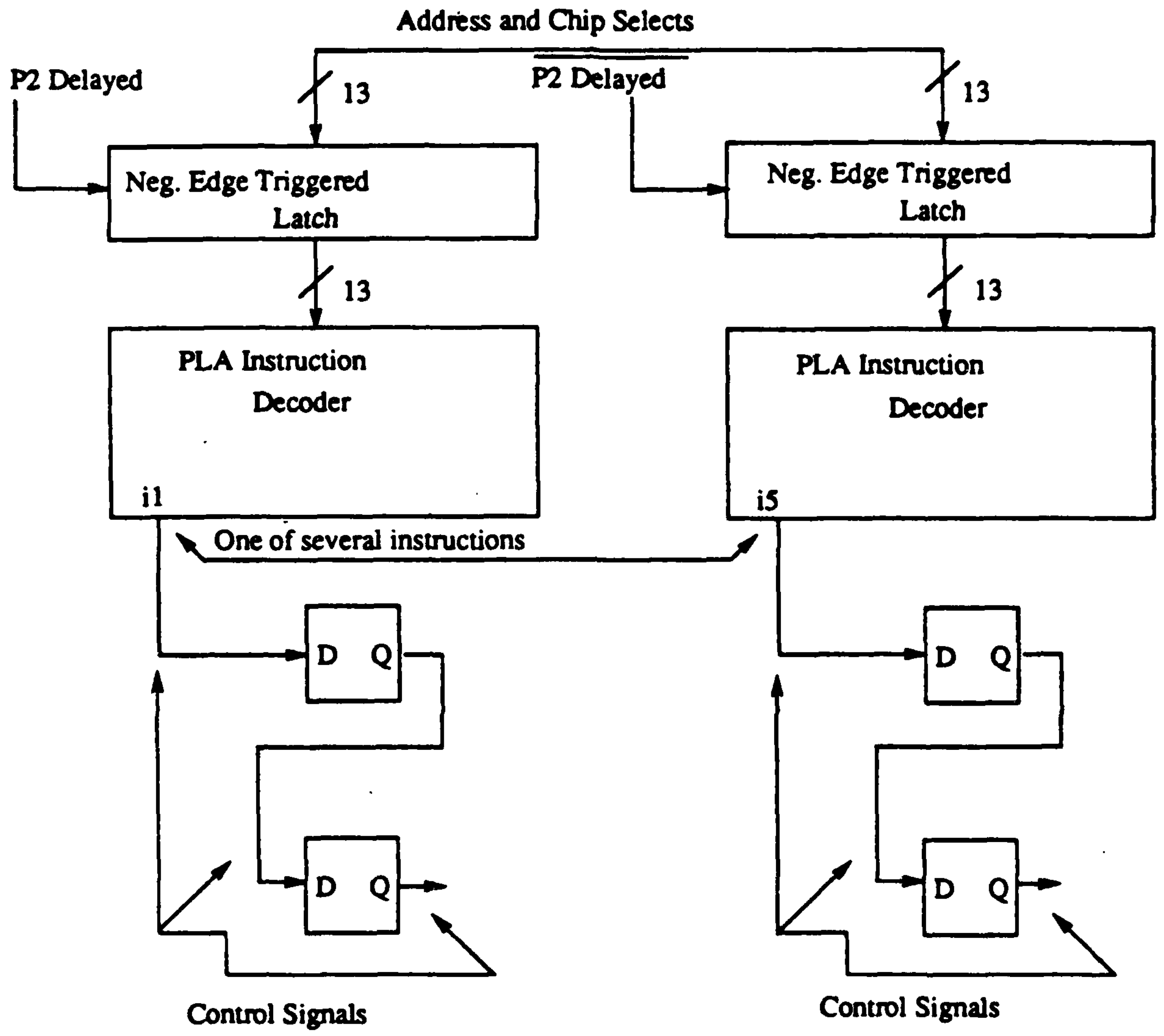


Figure 4.19

Block Diagram of the Controller

Instruction Set

I12	I11	I10	I09	I08	I07	I06	I05	I04	I03	I02	I01	I00	Instruction
1	1	1	1	1	1	1	1	1	1	0	0	0	Clear A, B, C
1	1	1	1	1	1	1	1	1	1	0	0	1	Load Status
1	1	1	1	1	1	1	1	1	1	0	1	0	Load B
1	1	1	1	1	1	1	1	1	1	0	1	1	Load A
1	1	1	1	1	1	1	1	1	1	1	0	0	Convert C
1	1	1	1	1	1	1	1	1	1	1	0	1	Show Status
1	1	1	1	1	1	1	1	1	1	1	1	0	Show Output
1	1	1	1	1	1	1	1	1	1	1	1	1	Clear C

Status Register

Stack Bus 31	Done Bit
Stack Bus 30	Interrupt Enable
Stack Bus 29	Sign Enable
Stack Bus 28 thru 0	Contents of the Output Register

Table 4.2

Instruction and Status Register Information

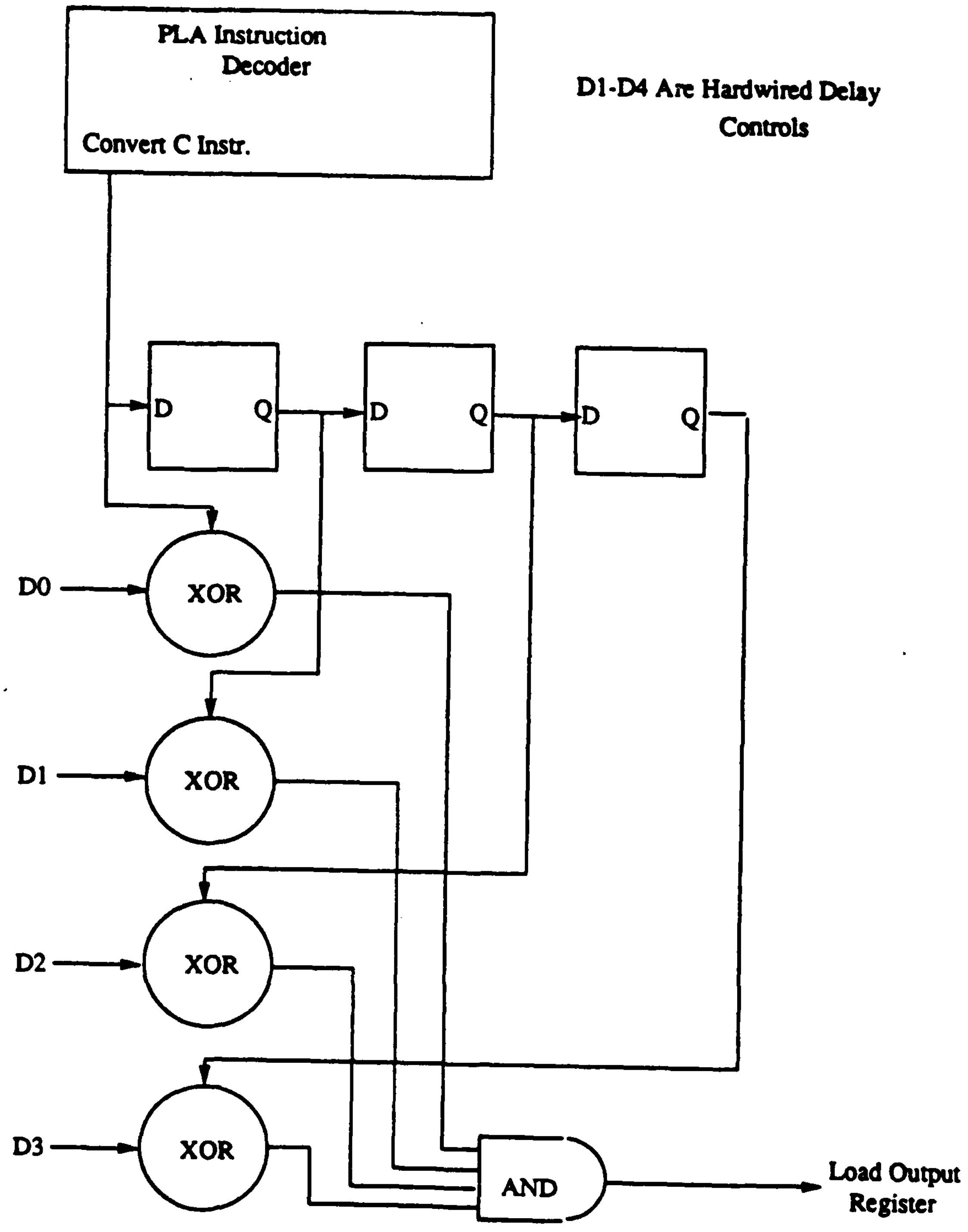


Figure 4.20

Delay Hardware

value of the B register while the C register is updated. This fact hold true if a conversion instruction is being waited upon. In this manner, the PRVP can be constantly fed useful data without delay.

The pin-outs for the PRVP are given in Fig. 4.21. The PRVP is mounted in a 84 pin grid array package.

4.4 TESTING

4.4.1 Problems

The PRVP was fabricated by MOSIS (Metal Oxide Semiconductor Implementation Service) in a 40 chip order using a CMOS 2-micron double-metal process. The PRVP was received eight weeks after the order was sent. This turn-around time was typical. Fig. 4.22 shows the die photograph obtained from MOSIS.

Much time was devoted to testing the PRVP for proper operation. From the test points, the PRVP seemed to be processing data as intended. However, strange results were coming out of the chip at the end of a test. There seemed to be a constant added to the correct result at times, with this problem being a systematic one. Using other test vectors, the results of the multiply-accumulate section appeared random.

Much time was devoted to finding the problem. Many theories explaining this weird behavior were tested, without success. From the test points, the correct data was being passed to the convert-back hardware. This was very misleading as to what the real problem was.

One test point was in error. This fact was not caught immediately, since this test point was initially confused with another one in the PRVP. This led quickly to the source of the problem.

It was discovered that one PLA in the convert-back block was not grounded. This fact was not caught by the simulator or by CSTAT. This problem was probably due to the use of global labels for the power and ground busses inside of this cell.

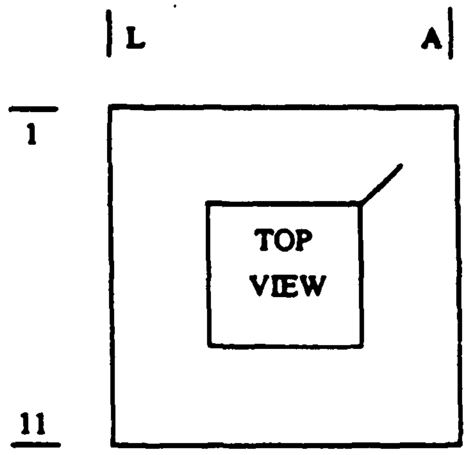
This fact did not block the verification of the design, however. Using a probe station designed to cut 50 square micron holes through the silicon oxide protective layer, a 5 square micron hole was cut through this protective layer so that this node could be reached by a probe.

A test fixture was constructed so that the tester and the probe station could be used simultaneously. Once this node was grounded using the probe station, the PRVP performed flawlessly. Many different vectors were sent through this chip to verify the blocks inside of the PRVP. The sign conversion sections and algorithms were especially tested rigorously, with four-quadrant behavior occurring at the correct times.

4.4.2 Speed and Power

Using the test points, it was a simple task to measure the delay through each major section of the PRVP. For this task, 12 chips out of the lot of forty were chosen at random. The average delay for a conversion from binary to residue was found to be 48.4 ns. (36 ns.) with a high of 54 ns. and a low of 44 ns. The delay for a multiply-accumulate operation averaged 59 ns. (37 ns.) with a high of 64 ns. and a low of 52 ns. The delay for the conversion from residue to binary was found to average 360 ns. (260 ns.) with a high of 405

Stack Bus 31 - k6 (MSB)
Stack Bus 30 - k5
Stack Bus 29 - l5
Stack Bus 28 - l6
Stack Bus 27 - l7
Stack Bus 26 - k7
Stack Bus 25 - j7
Stack Bus 24 - l8
Stack Bus 23 - l10
Stack Bus 22 - k9
Stack Bus 21 - l11
Stack Bus 20 - k10
Stack Bus 19 - j10
Stack Bus 18 - k11
Stack Bus 17 - j11
Stack Bus 16 - h10
Stack Bus 15 - h11
Stack Bus 14 - g9
Stack Bus 13 - g10
Stack Bus 12 - g11
Stack Bus 11 - f10
Stack Bus 10 - f9
Stack Bus 09 - e9
Stack Bus 08 - e11
Stack Bus 07 - e10
Stack Bus 06 - f11
Stack Bus 05 - d11
Stack Bus 04 - d10
Stack Bus 03 - c11
Stack Bus 02 - b11
Stack Bus 01 - c10
Stack Bus 00 - a11
Instruction Bus 12 -- a1 (MSB)
Instruction Bus 11 -- b3
Instruction Bus 10 -- a2
Instruction Bus 09 -- a3



Instruction Bus 08 -- b4
Instruction Bus 07 -- a4
Instruction Bus 06 -- a6
Instruction Bus 05 -- b5
Instruction Bus 04 -- a5
Instruction Bus 03 -- c5
Instruction Bus 02 -- c6
Instruction Bus 01 -- b6
Instruction Bus 00 -- a7
Vdd -- c2, l9, j6, l1
GND -- j2, j5, k8, b2
Interrupt Pin -- l4
Hardwire Status Register Select Pin -- k4
Interrupt Enable Pin -- l3
Sign Enable Pin -- l2
P2 Clock -- b7
P2 Delayed Clock -- c7
Reset (active low) -- a8
D0 -- b8
D1 -- a9
D2 -- a10
D3 -- b9
Test Points -- k1, j1, h2, h1, g3, g2, g1
No Connection -- k2, k3, b10, f1, f3, e3
No Connection -- e1, e2, f2, d1, d2, c1, b1

Figure 4.21

Pin Outs

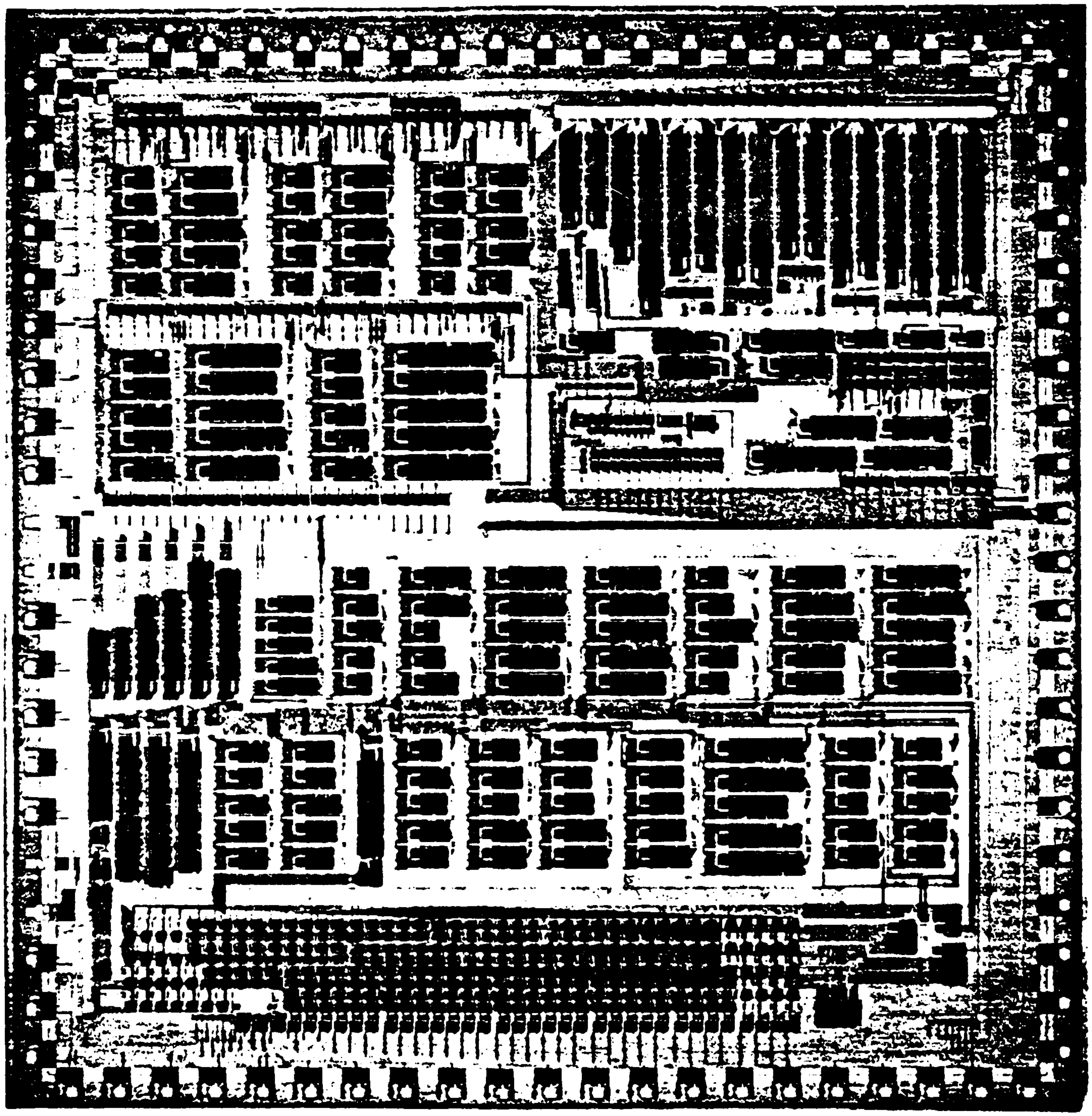


Figure 4.22

Photograph of the PRVP

ns. and a low of 300 ns. For all of these figures, the low figures were reasonably close to the expected delay from the SPLICE simulations (in parenthesis) of each block in question. The discrepancies can be attributed to modeling errors of the SPLICE package.

Using an ammeter on the test fixture, the static power dissipation was found to be just under 2 watts.

4.5 TASK III SUMMARY

The PRVP design reached its primary design goal. It is very fast. As a comparison, a TMS320C30 1–micron DSP chip can perform a multiply–accumulate operation on 24 bit integer values in 120 ns [36]. The PRVP can do the same operation with twice the technology size in half the time. Of course, the throughput of the PRVP will depend on the algorithm it implements. The actual throughput depends on how often a conversion from residue to binary is performed. If new operands are loaded as the PRVP is performing a conversion from residue to binary, a speed–up of approximately two can be expected over the TI chip.

The cost for this speed is hardware size. The PRVP is a large chip requiring a lot of area to implement. The design time for this large chip can be short, especially with good software tools such as automatic routers and cell placers. Using a RNS processor, fully–custom speed can be obtained using Semi–Custom design techniques. Therefore, the design cost/performance ratio is very low as compared to other chips that are typically the size of the PRVP.

Although two mathematical operations are performed in the PRVP, it does not provide an optimal application for a RNS processor except for certain algorithms. An ideal application for a RNS processor would be in high–order digital filters. In this application, many mathematical operations are performed on one data point, thus taking full advantage of the speed a RNS processor has to offer and minimizing the penalty for the conversion times. The PRVP represents a processor with a very wide data path for its speed and scope of application. A more typical application for a processor the speed of the PRVP would be a eight by eight multiply–accumulate with sixteen bit results due to the lack of availability of cheap sixteen–bit A/D converters at this time. Even today, eight–bit A/D converters with the speeds of the PRVP are very expensive. Therefore, wide processors such as the PRVP are ahead of other required system parts.

With this type of performance, the PRVP would seem to be an optimal design. However, several more design iterations could enhance the performance still further. One way that is currently being investigated is the use of multi–valued logic in RNS processors. This approach would cut the interconnect bandwidth and reduce the size of the logic required to implement an RNS processor.

However, an approach so advanced as multi–valued logic is not needed to enhance the performance of the PRVP. Throughout the design, the modulo–29 hardware proved to be significantly larger and slower than the hardware associated with the other moduli. One obvious way to speed up the PRVP is to choose another modulus other than 29 for the PRVP. Of course, in order to change from 29 to another modulus and still keep the interconnect bandwidth below 5 bits, a number which is not prime would have to be used. This may cause problems in the conversion from residue to binary, but these problems could be solved. This would easily improve the speed of the multiply–accumulate operation at least 10%, without any increase in hardware overhead (with an increase in design overhead).

The PLA's used throughout the design of the PRVP are far from ideal. They are pseudo–NMOS structures which draw much static power. They were used because they were the only static PLA's available for use with the MAGIC tool set. A set of dynamic PLA's are also available which do not draw any static power. However, these dynamic

PLA's were larger in size than the static PLA's. For the PRVP, power was traded for area on the chip. One important improvement in the design of another version of the PRVP would be a complete overhaul of the PLA tools which run with MAGIC. At the beginning of this project, this fact was discovered early on and an attempt was made to make sense of the C programming language source code which these PLA tools are written in. As with most of the MAGIC tools source code, it appeared to have many different authors each with a unique programming style which made the code impossible to follow. To improve these tools enough to make a large difference in the PRVP would be a separate subject for investigation and would have taken too much time. This improvement is certainly possible, and would likely cut the delay times in half.

ESPRESSO was a valuable tool for the minimization of the product terms each PLA implemented. However, this minimization was far from an optimal one. There are ways of helping ESPRESSO do its job which involve the phase of the PLA and the pairings of the input terms which could reduce the size of the existing PLA's up to 50% in size. This optimization would obviously cut the delay through the PRVP in half, while reducing its size markedly. This type of optimization is time consuming, and is large enough of a problem for another research topic.

One application of the PRVP which has not been discussed is the ability to place several PRVP's in parallel on one bus. A master processor can then utilize all the speed advantages of a RNS processor without paying much of a penalty for conversion times. As one PRVP is converting a number from residue to binary, the master processor can be feeding data to another PRVP. The instruction bus of the PRVP was designed with this application in mind so that each PRVP can easily be mapped into a different spot in the bus address space.

Section 5

CONCLUSIONS AND RECOMMENDATIONS

The Integrated Circuits for Avionics Program has successful by helped to establish the applicability of CMOS VLSI circuits to communications signal processing problems. Although the circuits were fabricated with 2- and 3-micron processes, the size and performance can be extrapolated to 1 micron or submicron processes which are currently available. Also the architectural approaches presented are generally independent of the fabrication technology.

5.1 CONCLUSIONS

5.1.1 Fast Arithmetic Circuits

Signal Processing problems almost always require the capability to do fast multiply, addition, and sometimes divide operations. For problems where pipelining and systolic array architectures can be applied (such as FIR filtering or convolution), fast and accurate arithmetic circuits can be realized. This is accomplished, as demonstrated in Task II, by incorporating pipelining in the parallel arithmetic circuits to a depth that permits clocking at rates consistent with registers and other clocked functions. The 3-micron circuit of Task II could be clocked in the 10–15 MHz range; this could be extrapolated to 30–40 MHz for 1 micron feature size. Note that the multiplier architecture presented in Task II can be pipelined so that its speed is independent of data word length (unlike the multiplier of Task I). Fast Multiplies and Divides in one clock cycle is a more difficult problem. Although not supported by this contract, Wright State has continued research to find hardware solutions to the fast multiply and divide in one clock (i.e., no pipelining). A CMOS circuit has been designed which performs a 16- X 16-bit booth multiply with 32-bit results or a 16- X 16-bit divide with 16-bit result and 16-bit remainder. The parallel divide circuit is especially unique in that previous approaches all relied upon multiple clock cycles.

5.1.2 Stringent Filtering Problem

Realizing circuits which provide very narrow bandwidths, small transition regions, and very high rejection in the stopband continues to be a difficult signal processing problem. Tasks I and II investigated methods for cascading VLSI circuits to effectively increase the order of thc filter to improve performance. CMOS circuits are limited to center frequencies in the tens of MHz even with pipelining and systolic array approaches. Increasing the order of the filter improves performance to a point where quantization, roundoff, and other such effects become limiting factors on stopband rejection and transition regions. Task I included investigating the improvements attained by using two different sampling rates for two portions of the cascaded systolic array. The performance can be significantly improved, but with the added complexity of multirate sampling.

5.1.3 Residue Number Processing

VLSI circuits make fast residue number processors a practical consideration. Task III presents a design for a high performance vector processor using residue number theory. Although the fabrication was done using 2–micron technology, it is clear that a significant price is paid for the hardware overhead to convert from binary to residue and from residue to binary. It is also clear that the parallel residue arithmetic can be performed very fast; even with 2–micron circuits the Task III processor is faster than most existing circuits. Maximum benefit can be achieved for an application where the converted binary data can be used by a number of residue processors, such as in a systolic array. Task III presents some new methods for realizing the conversion circuits in terms of mathematically partitioning the problem to use multiple PLA's and minimize the routing problems between PLA's. Still additional work is needed to reduce the hardware overhead and time delays associated with the conversion process for residue processors. Some work has been started at Wright State on this problem which uses same multivalue logic techniques to reduce the size of PLA's.

5.2 RECOMMENDATIONS FOR FURTHER WORK

5.2.1 Residue Number System VLSI Processors

Additional effort should be made to reduce the hardware overhead associated with binary to residue and residue to binary conversion. Some new techniques for efficiently performing the conversion process resulted from Task III, but additional work is warranted in such areas as (1) choosing the optimal set of moduli to minimize the conversion problem (2) minimizing the size of PLA's through multi–value logic techniques (3) improving existing tools to minimize the product terms resulting from a truth table for a PLA.

5.2.2 A Fast Parallel Divider

The residue number system approach does not support a fast parallel divide without carry as it does for multiply and add. Additional effort should be made to realize a VLSI circuit which performs a very fast divide without pipelining or multiple clock cycles. Encoding schemes and special hardware architectures could be investigated.

5.2.3 Highly Linear Analog Four Quadrant Multiplier

This program was aimed at fully digital VLSI circuits. Many signal processing applications could make use of a highly linear four quadrant multiplier. Such a device would facilitate monolithic version of a programmable sampled data analog signal processor. A fast, highly linear, four quadrant analog multiplier would also be a key element in new monolithic circuits for adaptive signal processors for a variety of communications applications.

5.2.4 Gallium Arsenide VLSI Technology

The Integrated Circuits for Avionics Programs investigated only silicon MOS circuits. Gallium Arsenide technology is attractive for high speed digital systems because of the higher mobility of the carriers in such devices. The MOSIS fabrication facility will soon support fabrication of GaAs designs, which would support university research into signal processing hardware using this technology. This would extend the frequency to the gigahertz range.

5.2.5 BiCMOS Circuits

An emerging technology is the combination of bipolar and MOS transistors in a common integrated circuit known as BiCMOS. The performance advantages compared to strictly bipolar or CMOS needs to be assessed for signal processing circuits. This can best be done through actual design, simulation, and fabrication of prototype circuits as was done under the Integrated Circuits for Avionics Program.

REFERENCES

[1] Rabiner, R. Lawrence, James H. McClellan, and Thomas W. Parks, "FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation," Proc. IEEE, Vol. 63, pp. 595–610, Apr. 1975.

[2] Freeny, Stanley L., "Special Purpose Hardware for Digital Filtering," Proc. IEEE, Vol. 63, pp. 663–648, Apr. 1975.

[3] Mead, Carver and Lynn Conway, "Introduction to VLSI Systems," Addison–Wesley Publishing Co., 1980.

[4] "MOSIS User's Manual," USC Informational Sciences Institute, 1985.

[5] Siferd, Raymond E., "Pipelining Highly Concurrent VLSI Processors To Meet Stringent Finite Impulse Filter Requirements," NAECON 1985, Vol. 1, pp. 2–7.

[6] Glasser, Lance A. and Daniel W. Dobberphul, "The Design and Analysis of VLSI Circuits," Addison–Wesley Publishing Co., 1985.

[7] Mavor, J., M. A. Jack, and P. B. Denyer, "Introduction to MOS LSI Design," Addison–Wesley Publishing Co., 1983.

[8] Birbal, Jerrold V. and Raymond E. Siferd, "Performance of a Custom VLSI Circuit for Programmable Signal Processing," NAECON 1986, Vol. 1, pp. 9–13.

[9] McClellan, J. H., T. W. Parks, and L. R. Rabiner, "FIR Linear Phase Filter Design Program," Programs for Digital Signal Processing, IEEE Acoustics, Speech, and Signal Processing Society, pp. 5–1.1 – 5–1.13, 1979.

[10] Vladimirescu, A., Kaihe Zhang, and A. R. Newton, D. O. Pederson, A. Sangiovanni–Vincentelli, "SPICE User's Guide," UW/NW VLSI Consortium VLSI Design Tools Reference Manual, 1985.

[11] Birbal, J. V., "Design of a Programmable Digital Filter Using nMOS VLSI Components," Interim Technical Report, Task I, Contract F33615–85–1718, Sept. 1986.

[12] Mukherjee, A., "Introduction to nMOS and CMOS VLSI System Design," Prentice–Hall, pp. 7–8, 1986.

[13] "Charger, 3 Micron Silicon–Gate CMOS/Bulk Cell Library," Volume 1–3, Revision 4, MOSIS Document 33A, June 1985.

[14] "Vale Reference Manual," Metheus–CV, Inc. 1985.

[15] Vladimirescu, A., K. Zang, A. R. Newton, D. O. Pederson, and A. Sangiouvanni–Vincentelli, "Spice User's Guide," University of California Berkeley, 1986.

[16] "Berkeley CAD Tools User's Manual," University of California Berkeley, 1986.

[17] Stanely, W. D., "Digital Signal Processing," Reston Publishing Company, Inc., 1975.

[18] Weste, N., and K. Eshraghian, "Principle of CMOS VLSI Design, A System Perspective," Addison-Wesley Publishing Co., pp. 331-333, pp. 344-348, 1985.

[19] Sweet, F. W., "Design of a Programmable Digital Filter Using CMOS VLSI Technology," Interim Technical Report, Task II, Contract F33615-85-1718, December 1987.

[20] Soderstrand, M. A., W. K. Jenkins, G. A. Jullien, and F. J. Taylor, "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing," IEEE Press, New York 1986.

[21] Szabo, N. S., and R. I. tanaka, "Residue Arithmetic and its Applications to Computer Technology," McGraw-Hill, 1967.

[22] Dixon, R. D. M. Calle, C. W. Longway, L. S. Peterson, and R. E. Siferd, "The SF1 Real Time Computer," Proceeding of the IEEE National Aerospace and Electronics Conference, 1988.

[23] Baraniecka, A., and G. A. Jullien, "On Decoding Techniques for Residue Number System Realizations of Digital Signal Processing Hardware," IEEE Trans. Circuits Syst., Vol. CAS-25, pp. 935-936, Nov. 1978.

[24] Soderstrand, M. A., "A New Hardware Implementation of Modulo Adders for Residue Number Systems," Proceedings fo the 26th Midwest Symposium of Circuits and Systems, 1983.

[25] Soderstrand, M. A., and E. L. Fields, "Multipliers for Residue-Number Arithmetic Digital Filters," Elect. Lett., Vol. 13, No. 6, pp. 164-166, Mar. 17, 1977.

[26] Taylor, F. J., "Large Moduli Multipliers for Signal Processing," IEEE Trans. Circuits Syst., Vol. CAS-28, pp. 731-736, July 1981.

[27] Kinoshita, E., H. Kosako, and Y. Kojima, "General Division in the Symmetric Residue Number System," IEEE Trans. Comput., Vol. C-22, pp. 134-142, Feb. 1973.

[28] Bayoumi, M. A., G. A. Jullien, and W. C. Miller, "Models for VLSI Implementation of Residue Number System Arithmetic Modules," Proceedings of the 6th Symposium on Computer Arithmetic, pp. 174-182, 1983.

[29] Bayoumi, M. A., G. A. Jullien, and W. C. Miller, "A Look-Up Table VLSI Design Methodology for RNS Structures Used in DSP Applications," IEEE Trans. Circuits Syst., vol. CAS-34, pp. 604-616, Jun. 1987.

[30] Bayoumi, M. A., G. A. Jullien, and W. C. Miller, "A VLSI Implementation of Residue Adders," IEEE Trans. Circuits Syst., Vol. CAS-34, Mar. 1987.

[31] Thomas, R., and J. Yates, "A User's Guide to the UNIX System," McGraw-Hill, 1985.

[32] "1986 VLSI Tools: Still More Works by the Original Artists," Computer Science Division of the University of California at Berkeley.

[33] Kung, S. Y., H. J. Whitehouse, and T. Kailath, "VLSI and Modern Signal Processing," Prentic–Hall 1985.

[34] "Third–Generation TMS320 User's Guide," Texas Instruments Inc., 1988.

[35] Hohne, R. A., and R. E. Siferd, "A Programmable High Performance Processor Using the Residue Number System and CMOS VLSI Technology," IEEE NAECON, Vol.1, pp. 41–43, May 1986.

[36] Hohne, R. A., "A High–Performance Vector Processor Using Residue Number Theory, Pipelining, and VLSI Technology," Interim Technical Report, Task III, Contract F33615–85–1718, June 1989.